COMPUTER
System and Data Analysis

S. SRINIVASAN

VISIT…

LANZAROTE
Caliente.COM

COMPUTER SYSTEM
AND
DATA ANALYSIS

S SRINIVASAN

CENTRAL

Our
Publications on ENGINEERING

Design and Construction of Highway Bridges	*K. S. Rakshit*
Principles of Machine Tools	*Sen & Bhattacharyya*
Principles of Metal Cutting	*Amitabha Bhattacharyya*
Technology of Machining Systems	*Mullick & Bhattacharyya*
Robotic Applications in India	*Amitabha Bhattacharyya*
Fortran Based Algorithm for Engineering Computations	*Ranesh Sircar*
Statistical Techniques and Applications for Engineers	*Ranesh Sircar*
Computer Aided Design of Chemical Process Equipments	*B. C. Bhattacharyya & C. M. Narayanan*
Computer Numerical Control (CNC) Machines	*P. Radhakrishnan*
Material Science	*Panigrahi & Dutta*

COMPUTER SYSTEM
and
DATA ANALYSIS

S SRINIVASAN
Professor, Dept of Computer Science and Engineering
R V College of Engineering, Bangalore

NEW CENTRAL BOOK AGENCY (P) LTD
8/1 Chintamoni Das Lane, Kolkata 700 009
INDIA

COMPUTER SYSTEM
and
DATA ANALYSIS

First Published: September 1993
With Additions: March 2008
Revised Edition: 2020

© Copyright reserved by the Author
Publication, Distribution, and Promotion Rights reserved by the Publisher

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic, mechanical, photo-copying, recording or any information storage and retrieval system, without the prior written permission of the Publisher.

Publisher and Typesetter
NEW CENTRAL BOOK AGENCY (P) LTD
8/1 Chintamoni Das Lane, Kolkata 700 009

Printer
NEW CENTRAL BOOK AGENCY (P) LTD
Web-Offset Division, Dhulagarh, Sankrial, Howrah

Cover Designer
Dulal Halder

Cover Printer
BISWANATH STUDIO
3 Garanhatta Street, Kolkata 700 006

Project Supervisor
Abhijit Sen

Diagrams
Narayan Sur

ISBN: 978-1-64725-149-9

Preface

Part A of the book deals with the computer basics. It gives the introductory details about the HARDWARE of the present-day Electronic Digital Computer. Incidentally, it contains a discussion on the various Number Systems related to computer processing.

Part B introduces the reader to the computer software. It contains a fairly detailed discussion on Data Processing on Computer with FORTRAN. Special care has been taken that the Examples and the Exercises given in this section will cover the majority of the problems that a student in Science or Engineering will come across during his or her studies.

Part C deals with the computer oriented Numerical Methods. Here also the numerical methods for solving the problems—that a Science or Engineering student may come across—are discussed with illustrating examples.

Part D provides brief details of some of the popular Programming Languages, regarding their facilities and limitations. FORTRAN 77 is dealt briefly in this part since FORTRAN IV has been covered in detail in Part B.

This book can be used by the interested learners who are eager to have a factual and fundamental knowledge about the computer hardware and software, in as much as the book is written assuming no prior knowledge on the subject on the part of the student or the reader.

The author will be thankful for constructive suggestions from the users of this book.

S SRINIVASAN

COMPUTER SYSTEM
AND
DATA ANALYSIS

SECTION A

NON-DIPLOMA STREAM

Part A

Computer Basics

Organization of Digital Computer. Block Diagram, Major Elements, Numbersystems and Codes, Radix Conversion, Input-Output Devices, Storage Devices, Arithmetic Unit and Control.

Part B

Elements of Programming

Introduction. Constants. Variables and Expressions. Arithmetic Statement. Functions. Input and Output Statements. Transfer of Control G0 T0, IF Statements. Subscriptive Variables. Use of D0 Statements. Formats. Functions and Specification Statements. Use of Subroutines. Programming Exercises.

Part C

Introductory Numerical Analysis

Approximation, Error. Significant Figures. Eigen Values and Eigen Vectors. Matrix Inversion. Numerical Integration and Numerical Solution of Ordinary Differential Equation.

Part D

Computer Languages

Distinctive Features of FORTRAN, ALGOL, COBOL, PL/1. Compliers and Translators. Multi-programming. Multi-processing and Time Sharing.

Contents

Pages

Chapter One

Organisation of Digital Computer 3 – 8
1.1 Introduction 3
1.2 Digital Computer Fundamentals 3
1.3 Organisation of a Digital Computer 4
1.4 Data Processing in Digital Computers 5
1.5 Types of Digital Computers (Classification according to Application) 6
1.6 Classification of Digital Computers according to their size and performance 6
1.7 Characteristics of Digital Computers 7
Summary 7
Exercise/Questions 7
for NOTES 8

Chapter Two

Number Systems and Radix Conversions 9 – 18
2.1 Introduction 9
2.2 Number Systems 9
2.3 Positional Systems 9
2.4 Different Number Systems 10
2.5 Conversion of Decimal Number into other Systems 10
2.6 Conversion of a Number from Other Systems to the Decimal System 13
2.7 Conversion among Hexadecimal, Octal and Binary Systems 15
2.8 Computer Arithmetic 17
Summary 17
Exercise/Questions 17
for NOTES 18

Chapter Three

Numeric Codes 19 – 30

3.1 Introduction 19
3.2 Classification of the Numeric Codes 19
3.3 Weighted Codes 20
3.4 Non-Weighted Codes 20
3.5 Alphameric Codes 26
3.6 Display Codes 28
Summary 29
Exercise/Questions 30
for NOTES 30

Chapter Four

Arithmetic Unit 31 – 47

4.1 Introduction 31
4.2 Arithmetic Operations 32
4.3 Binary Subtraction 34
4.4 Multiplication by the ALU 42
4.5 Division in Computers 43
4.6 Logical Operations 46
Summary 46
Exercise/Questions 47
for NOTES 47

Chapter Five

The Control Unit 48 – 51

5.1 Introduction 48
5.2 Processing Operation 49
5.3 Types of Control 49
Summary 51
Exercise/Questions 51
for NOTES 51

Pages

Chapter Six

Input and Output Devices 52 – 64

6.1 Introduction 52

6.2 Input Devices 52

6.3 Output Devices 60

Summary 63

Exercise/Questions 63

for NOTES 64

Chapter Seven

Memory Units 65 – 69

7.1 Introduction 65

7.2 Memory Address 65

7.3 Bit, Byte, Word 65

7.4 Types of Memory 66

7.5 Secondary Memory Devices 68

Summary 68

Exercise/Questions 69

for NOTES 69

Chapter Eight

Introduction to Programming 73 – 82

8.1 Introduction 73

8.2 Definition of Data Processing and Data 73

8.3 Major Steps in Data Processing 74

Summary 81

Exercise/Questions 81

for NOTES 82

Chapter Nine

Constants and Variables in Fortran Program 83 – 89

9.1 FORTRAN Language 83

9.2 FORTRAN Program 83

Pages

9.3 Elements of FORTRAN Language 84
9.4 Constants and Variables 85
Summary 88
Exercise/Questions 89
for NOTES 89

Chapter Ten

Input-Output Statements 90 – 106

10.1 Character Set 90
10.2 Types of FORTRAN Statements 90
10.3 I/O Statements 91
10.4 Output Statement 98
10.5 End of File Checking Incorporated Read Statement 104
Summary 104
Exercise/Questions 105
for NOTES 106

Chapter Eleven

Arithmetic Statements and Library Functions 107 – 118

11.1 Processing Statements 107
11.2 Library Functions 112
11.3 Usefulness of Arithmetic Statements 116
Summary 117
Exercise/Questions 117
for NOTES 118

Chapter Twelve

Control Statements 119 – 131

12.1 Classification of Control Statements 119
12.2 Program Control Statements 119
12.3 Transfer of Control or Branch Statements 119

12.4 Conditional Transfer of Control Statements 120
12.5 Computed GOTO Statement 127
12.6 Assigned GOTO Statement 129
Summary 130
Exercise/Questions 130
for NOTES 131

Chapter Thirteen

Loop Statements: Do Statement and Subscripted Variables and Processor Control Statements 132 – 146
13.1 Introduction 132
13.2 Processor Control Statements 143
Summary 145
Exercise/Questions 145
for NOTES 146

Chapter Fourteen

Sub-programs (sub-routines) and Specification Statements 147 – 165
14.1 Introduction 147
14.2 Specification Statements 160
14.3 Order of Statements in a Program 164
Summary 164
Exercise/Questions 165
for NOTES 165

Chapter Fifteen

Additional Programming Exercises 166 – 167
Exercises/Questions 166
for NOTES 167

Pages

Chapter Sixteen

Approximation, Significant Digits and Errors 171 – 177

16.1 Computer Oriented Numerical Methods 171
16.2 Approximation 171
16.3 Significant Digits 171
16.4 Errors 172
16.5 Measure of Errors 173
16.6 Propagation of Errors 173
16.7 Series Approximation 176
Summary 176
Exercise/Questions 177
for NOTES 177

Chapter Seventeen

Linear, Algebraic and Transcendental Equations 178 – 195

17.1 Introduction 178
17.2 Linear Equations 178
17.3 Algebraic Equations 179
17.4 Transcendental Equations 179
17.5 Numerical Methods 179
17.6 Conclusion 193
Summary 194
Exercise/Questions 194
for NOTES 195

Chapter Eighteen

Simultaneous Algebraic Equations—Matrix Inversion 196 – 212

18.1 Introduction 196
18.2 Simultaneous Algebraic Equations 196
18.3 Indirect Methods of Solution to Non-Homogeneous Linear Algebraic Equations 208
Summary 211
Exercise/Questions 211
for NOTES 212

Page

Chapter Nineteen

Eigen Values and Eigen Vectors 213 – 222

19.1 Solution to Homogeneous Simultaneous Linear Algebraic Equations 213

19.2 Eigen Value Technique 214

19.3 Computer Determination of Eigen Values and Vectors 216

Summary 221

Exercise/Questions 222

for NOTES 222

Chapter Twenty

Numerical Integration 223 – 229

20.1 Introduction 223

20.2 The Trapezoid Rule 224

20.3 Simpson's Rule 226

Summary 229

Exercise/Questions 229

for NOTES 229

Chapter Twentyone

Numerical Solution to Ordinary Differential Equations 230 – 241

21.1 Introduction 230

21.2 Initial Value Problems 231

21.3 Picard's Method 231

21.4 Euler's Method 232

21.5 Modified Euler's Method 233

21.6 Second Order Runge-Kutta Method 235

21.7 Runge-Kutta Fourth Order Method 237

21.8 Taylor's Series Method 237

21.9 Predictor-Corrector Methods 238

Summary 240

Exercise/Questions 241

for NOTES 241

Page

Chapter Twentytwo

Overview of Computer Languages and Their Distinctive Features 245 – 250

22.1 Introduction 245
22.2 Types of Semantics 246
22.3 Types of Programming Languages 246
22.4 Machine Language 246
22.5 Assembly Language 247
22.6 High Level Languages 247
22.7 Classification of High Level Languages 247
22.8 Features of High Level Languages 247
22.9 Advantages in High Level Languages 248
22.10 Structure of a Program in High Level Language 248
22.11 The Hierarchical Structure of a Program 248
22.12 Distinctive Features of Programming Languages 249
Summary 249
Exercise/Questions 250
for NOTES 250

Chapter Twentythree

FORTRAN 251 – 255

23.1 Introduction 251
23.2 Character Set 251
23.3 Data Types 251
23.4 Types of Statements 252
23.5 Sequence Control Statements 252
23.6 Iteration Statements 253
23.7 Sub-Programs and Library Functions 253
23.8 Character Strings 254
23.9 File Processing 254
Summary 255
Exercise/Questions 255
for NOTES 255

Page

Chapter Twentyfour

ALGOL 256 – 261

24.1 Introduction 256
24.2 The Character Set 256
24.3 Keywords 257
24.4 Constants and Variables 257
24.5 I/O Statements 257
24.6 Arithmetic Statements 258
24.7 Relational and Boolean Operators 258
24.8 Control Statements 259
24.9 Loop Statements 259
24.10 Procedures 259
24.11 Library Functions 260
24.12 Structure of ALGOL Program 260
24.13 Conclusion 261
Summary 261
Exercise/Questions 261
for NOTES 261

Chapter Twentyfive

COBOL 262 – 272

25.1 Introduction 262
25.2 Structure of Cobol Program 262
25.3 Types of Sentences 262
25.4 Key-Words and Optional Words 263
25.5 Classification of Words 263
25.6 Coding COBOL Program 267
25.7 A Specimen Program in COBOL 268
Summary 269
Exercise/Questions 269
for NOTES 269
COBOL Program Sheets 270 – 272

Page

Chapter Twentysix

PL/1 273 – 279

26.1 Introduction 273
26.2 Character Set 273
26.3 Types of Data 274
26.4 I/O Operations 274
26.5 Arithmetic And Relational Operators 275
26.6 Control-Statements 275
26.7 Library Functions 276
26.8 Sub-Programs/Procedures 277
26.9 Program Structure 277
26.10 Specimen Program 278
26.11 Conclusion 278
Summary 278
Exercise/Questions 278
for NOTES 279

Chapter Twentyseven

Translators and Compilers 280 – 284

27.1 Introduction 280
27.2 Translators 281
27.3 Compilers 281
27.4 Phases in Compilation 281
Summary 283
Exercise/Questions 284
for NOTES 284

Chapter Twentyeight

Multi Processing, Multi Programming and Time Sharing 285 – 287

28.1 Introduction 285
28.2 Multi-Tasking 285
28.3 Other Definitions 286
Summary 287
Exercise/Questions 287
for NOTES 287

Part A

Computer Basics

Chapter One Organisation of Digital Computer *3 – 8*

Chapter Two Number Systems and RADIX Conversions *9 – 18*

Chapter Three Numeric Codes *19 – 30*

Chapter Four Arithmetic Unit *31 – 47*

Chapter Five The Control Unit *48 – 51*

Chapter Six Input and Output Devices *52 – 64*

Chapter Seven Memory Units *65 – 69*

"This page is Intentionally Left Blank"

Chapter One

Organisation of Digital Computer

1.1 INTRODUCTION

A COMPUTER is a device which is designed to perform Arithmetic and Logical operations.

The computers are of 3 types, namely—

(i) Analog Computers,
(ii) Digital Computers, and
(iii) Hybrid Computers.

An Analog Computer can be said to be a measuring device. The input to it will be analogous to the physical data that is analysed. As such it serves as a model for a physical system and simulates it.

A Digital Computer takes in Numerical Data and, making use of controlled Arithmetic, performs Arithmetic and Logical operations on them.

A Hybrid Computer is a combination of Analog and Digital Computers.

1.2 DIGITAL COMPUTER FUNDAMENTALS

The present-day Digital Computer is an electronic device. It can be termed as a Counting device which takes in a numeric data, processes it and gives out a result which is also numeric. It may be noted here that the present-day Digital Computer can take in and process alphameric or string data also.

The details given as input to the Digital Computer are called DATA. (It may be noted that the word data is both used as singular and plural.) The output result from the Digital Computer is often called INFORMATION. The Digital Computer takes in an input DATA, processes it and gives out the output information or the result.

Processing involves—(a) Arithmetic and (b) Logical operations. The Arithmetic Operations are—(i) Addition, (ii) Subtraction, (iii) Multiplication and (iv) Division. Using successive multiplications, Exponentiation (raising a number to some power) can also be performed in a Digital Computer.

The Logical Operations that can be performed in a Digital Computer are, checking whether a variable, say, A is equal to or greater than or less than another variable, say, B or a constant.

To get the desired result from the given data, the user should give detailed instructions in proper sequence to the computer for performing the processing. This set of instructions is called PROGRAM. Programs have to be written in special languages that are known as Programming Languages.

1.3 ORGANISATION OF A DIGITAL COMPUTER

The Organisation or the Architecture of a Digital Computer is shown in the Block Diagram 1.1.

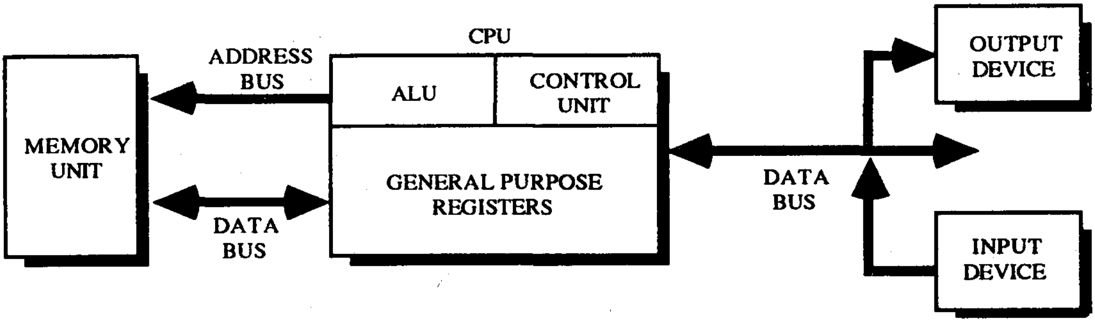


Fig 1.1 Block Diagram of a Digital Computer

As shown in the diagram, a Digital Computer System contains—

(i) A Central Processing Unit (CPU),

(ii) One or more Input Device/s,

(iii) One or more Output Device/s, and

(iv) A Storage unit (MEMORY).

The Central Processing Unit is also called the CPU or the PROCESSOR. It is made up of 3 main sections, namely—(a) The Arithmetic Logic Unit (ALU), (b) The Control Unit, and (c) A few General Purpose Registers.

The ALU is to perform the Arithmetic and Logical operations.

The Control Unit is to decode the instructions in the user's program and send signals to the various devices in the computer system to perform the various processing operations.

The General Purpose Registers are useful for storing temporarily the partial results.

The purpose of the Input Devices is to take in the input Data and the Program into the computer system. This Input operation is called a READ operation.

The input Device may be—(a) A Key-Board, (b) A Card-Reader, (c) A Paper-Tape-Reader, (d) A Magnetic Tape Unit, (e) A Magnetic Disk Unit or (f) A Floppy Disk. The Key-Board and Floppy-Disk are used as input devices in most of the Computer Systems nowadays.

The Output devices are intended to display the information or the results. The Output operation is often called 'WRITE' operation.

The various output devices are—(a) A Card-Punch, (b) A Paper-Tape-Punch, (c) A Magnetic Tape, (d) A Magnetic Disk, (e) A Floppy Disk, (f) Screen or (g) A Printer. The Screen and the Printer are the mostly used output devices.

It is not that a computer can use only one input device and one output device at a time. Big computers like the I.B.M–370 can simultaneously operate on a number of input and output devices at a time. The input-output devices are often referred as I.O. Devices.

The storage unit is also called the MEMORY. A MEMORY or storage unit is essential in a computer system for the following purposes—

(i) To store control Programs or the Operating System (O.S.). O.S. is a program which controls all the operations in the computer system.

(ii) To store Library Routines. The digital computer is essentially a counting machine. As such, it cannot evaluate logarithms and trigonometric functions like sin θ. So sub-programs, known as FUNCTIONS, are prepared by the manufacturer to evaluate such functions using some numerical methods and they are permanently stored in the memory. These are called Library Functions. Whenever an expression like log(x) or sin(y) are used in the user's program, the processor will execute the corresponding Library Function and compute the result.

(iii) To store the input Data.

(iv) To store the partial or intermediate results.

(v) To store the USER'S program. The user's program is stored in the memory and is executed on the input records, one by one. This method of processing the input records one by one by the computer using the program stored in its memory is called 'STORED PROGRAM CONCEPT'.

The MEMORY or the Storage device is composed of a number of locations. (eg,) A MEMORY of size 8 K (8 kilo) will have 8192 locations which will be referred to or addressed as 0000 to 8191. Each location will contain 8 bits. (Bit is the abbreviation for BINARY DIGIT). These 8 bits are used to store any character in the Binary form. A group of such 8 bits is referred as a BYTE. In the Binary form, any character will be represented by a group of 0's and 1's. A character is any alphabet A to Z, or any digit 0 to 9 or any symbol like +, −, =, etc.

The total number of locations in any memory unit will be in the order of 2^n where n is an integer and will be equal to the number of lines or bits in what is known as ADDRESS BUS. The address bus is used by the processor to access any location in the memory unit or an I.O. device.

To take out data or send data to either the memory or the I.O. device, the processor uses the DATA BUS. The number of lines in the data bus depends upon the WORD-SIZE of the computer. The word is an ordered set of characters which can be transferred in a computer circuit as a single unit. The number of bits in such a word is called the word-size. In a 16 bit computer, for example, the word size is 16 bits or 2 bytes, ie, 2 bytes or 2 characters can be transferred at a time.

1.4 DATA PROCESSING IN DIGITAL COMPUTERS

The input data to the digital computer is normally of 3 forms, namely—(i) File, (ii) Record, and (iii) Field.

A record contains the details related to one person or one material, or one set of readings from an experiment or a survey. For example, the marks-card of a student is a record. A File contains the records of same type or transaction. For example, the set of marks-cards of the students of the same class will constitute a file.

Each record contains one or more fields. For example, the register-number, name, class, centre, marks in subject 1, marks in subject 2 etc. in a marks record will each be a Field.

The computer takes up a file and processes record by record. In each record, the processing is done on its fields.

Processing data using a Digital Computer is of 3 major steps, namely—(i) Data preparation, (ii) Program preparation, and (iii) Compiling and Running the program. These will be described in detail in the chapters under Part B.

1.5 TYPES OF DIGITAL COMPUTERS (Classification according to Application)

The Digital Computers are classified into 3 types, namely—(a) Computers for processing scientific and technical data, (b) Computers for business data processing, and (c) Versatile computers, depending upon their applications.

The computer for processing scientific and technical data should be capable of performing intricate and advanced mathematical operations. It should be able to give precise results with a high degree of accuracy.

The computer for processing Business data should be able to handle a number of files simultaneously and each of these files may contain a large number of records.

Versatile computers are those that can be used to process both technical and business data. Most of the present-day digital computers are versatile computers.

The versatile computers themselves are of 2 types, namely—(i) General purpose computers, and (ii) Special purpose computers.

The general purpose computers are useful in performing a variety of jobs in scientific and technical and business/commercial fields.

The special purpose computers are designed for a particular type of application such as—(i) Numerical control of machine tools, (ii) Electronic switching system in telephones, and (iii) Navigational devices in the spacecrafts and satellites, etc.

1.6 CLASSIFICATION OF DIGITAL COMPUTERS ACCORDING TO THEIR SIZE AND PERFORMANCE

Depending on the performance and size, the digital computers can be classified as—(i) Microcomputers, (ii) Minicomputers, (iii) Midicomputers, (iv) Large Computers, and (v) Super Computers.

The Desk Top Computers or the Personal Computers (PC's) fall under the category of microcomputers. These computers use a Micro-Processor for the CPU. They are mostly provided with a memory of size 64 Kilo Bytes (64 K).

Digital Equipment Corporation's (DEC) PDP-11 is an example of a Mini Computer. It has a word-size of 2 bytes and uses a memory unit of size 256 K.

Midi Computers have a word-size of 24 bits (3 bytes) and use a memory of size 1 MB (Mega Byte).

IBM 360 and 370 are examples of large computers. They have a word-size of 4 bytes and use a large memory unit, say, of size 16MB. They have a number of advanced architectural features such as separate I/O Processors, Floating point Arithmetic processors, etc. These computers are also known as MAIN FRAME computers.

The supercomputers have many processors operating in parallel. They have all the advanced architectural features as the Main Frames and their memory size may go up to 10 Giga bytes, eg, Cray Corporation's XMP COMPUTER.

1.7 CHARACTERISTICS OF DIGITAL COMPUTERS

A Digital Computer of present generation has the following characteristics—

(i) It is capable of executing the instructions stored in its memory.

(ii) It is versatile, in the sense that it can perform commercial jobs such as preparation of pay-bill, inventory report etc., and also solve technical problems such as stress-analysis, network analysis etc.

(iii) It gives the results with high precision and the error-rate, as compared to humans, is extremely low.

(iv) It is very fast. Even complicated and large calculations are performed in a digital computer not only accurately, but also at a very high speed.

Summary

A Digital computer is a device that performs Arithmetic and Logic Operations. The present-day Digital computer is an Electronic device. It is used to process both technical and business data.

The main component of the computer is its CPU. It contains an Arithmetic Logic Unit, a set of General Purpose Registers and a Control Unit.

A computer system should have one or more input devices to take in the data and the program. So also it should have one or more output devices to display the result or the information. It should also have a memory unit or storage device to store the Operating System, user's programs and the data.

Exercise/Questions

1.1 What is a computer? What are the three types of computers?

1.2 Define a 'digital computer'. What are its characteristics?

1.3 Describe with a block diagram the major elements in a computer system. Explain the function of each element.

1.4 Define the terms—(i) CPU, (ii) ALU, (iii) Data, (iv) Information, (v) Operating System, (vi) Program, (vii) Library Functions, and (viii) Main-Frame.

1.5 How do you classify the Digital Computers—(a) according to their applications, and (b) according to their size?

1.6 The XMP Super-Computer of Cray Corporation has a word-size of 64 bits. Express this in terms of bytes.

1.7 The Intel 8086 Micro-Processor has a word-size of 2 bytes. Express this in bits.

1.8 Explain what is meant by 'Stored Program Concept'?

1.9 Explain the two types of versatile computers.

for NOTES

Chapter Two

Number Systems and RADIX Conversions

2.1 INTRODUCTION

The numeric data to be processed by the computer will be recorded by the user in the Decimal form. This data is stored in the memory of the computer mostly in the Hexadecimal form. In the earlier versions of some Micro-Processors it was stored in the Octal form. Processing of data in the computer is done with the numbers in the Binary form. As such, this chapter discusses these Number Systems as well as the method of conversion of a number from one system to another.

2.2 NUMBER SYSTEMS

A Number System consists of a set of Symbols and Rules for purpose of representing any number.

There are 2 types of number systems, namely—(i) Positional Systems, and (ii) Non-Positional Systems. Of these the Positional Systems are the ones that are widely used.

2.3 POSITIONAL SYSTEMS

The Positional Systems follow the following Rules—

(i) Each system has a RADIX which is also called BASE. (eg,) The Radix in the Decimal system is 10.

(ii) Each system has a definite number of coefficients. Their number will be equal to the Radix itself. (eg,) In the Decimal system there are 10 coefficients, namely—0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. These coefficients are also called Discrete Characters.

(iii) The maximum value of a coefficient in the system will be one less than the Radix. (eg,) In the Decimal system, the maximum value of the coefficient is 9, which is 1 less than the Radix, namely—10.

(iv) In any number system, a number is expressed as a sequence of digits (eg,) 4673. The sequence is also called a STRING.

(v) Each digit position in the string has a value. This value is also known as Weight. The weight will be equal to Radix^n. The value of n will be 0 for the rightmost digit and will increase by 1 for the digit to the left. For example, in the Decimal system, the weight of the rightmost digit is 10^0 (units). The weight of the next digit to the left is 10^1 (tens), the weight of the next one to the left is 10^2 (hundreds) and so on. For example, the value of 4673 in the decimal system will be $3 \times 10^0 + 7 \times 10^1 + 6 \times 10^2 + 4 \times 10^3$.

EXAMPLE 2.1 Give the Decimal value of 576 which is in the Octal system.

Solution :

5 7 6

$$\begin{array}{rcl} 6 \times 8^0 = 6 \times 1 & = & 6 \\ 7 \times 8^1 = 7 \times 8 & = & 56 \\ 5 \times 8^2 = 5 \times 64 & = & \underline{320} \\ & & 382 \end{array}$$

$(576)_8 = (382)_{10}$

2.3.1 Alternate Method of Evaluation

The decimal value of any number in any other system may also be evaluated by the use of the following formula—

$$N = S \cdot B^m \left(\sum_{k=1}^{m} n_k \cdot B^{-k} \right)$$

where N is the decimal value; S = sign; B = Radix; m = number of digits, and n_k = digit in the kth position.

EXAMPLE 2.2 Evaluate the decimal value of $-(576)_8$ using the above formula.

Solution : $B = 8$ and $m = 3$ $S = -$ (NEGATIVE SIGN). So $N = -8^3(5 \times 8^{-1} + 7 \times 8^{-2} + 6 \times 8^{-3})$ $= -(5 \times 8^2 + 7 \times 8 + 6 \times 8^0) = -(382)_{10}$.

2.4 DIFFERENT NUMBER SYSTEMS

As already stated, the different number systems that are connected with computers are— (i) the Decimal system, (ii) the Binary system, (iii) the Hexadecimal system and (iv) the Octal system.

In the Decimal system, the Radix is 10. The coefficients are 0, 1, 2, 3, etc 8, 9. The maximum value of the coefficient is 9 (Radix−1) and the number of coefficients 10.

In the Binary system, the Radix or Base is 2. The coefficients or discrete characters are 0 and 1 only. The maximum value of the coefficient is 1 and the number of coefficients 2, which is equal to the Radix.

In the Hexadecimal system, the Radix is 16. So, this system has 16 coefficients, namely, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F (the alphabets A to F are used to represent the numbers 10 to 15 respectively as single digits). The maximum value of a coefficient is F, that is, 15.

In the Octal system, the Radix is 8. The eight coefficients in this number system are 0, 1, 2, 3, 4, 5, 6 and 7, the maximum value of a coefficient being 7.

2.5 CONVERSION OF DECIMAL NUMBER INTO OTHER SYSTEMS

For converting a Decimal number into another system the following rules are to be followed—

(i) Successively divide the integer portion of the Decimal number by the new Radix until the Quotient becomes 0. Write down the remainders in the various stages in the reverse order. This will give the integer portion of the number to the new base.

(ii) To convert the fractional portion of a decimal number to another system: Multiply it successively by the new Radix until the fractional part of the product becomes more or less 0. Then write down the integer portion obtained in each stage in order. The result preceded by a decimal-point will give the value of the decimal fraction to the new Radix.

The following examples will illustrate the above procedures.

EXAMPLE 2.3 Convert 37 from the decimal system to the Binary system.

Solution :

$$\begin{array}{r|l l l}
2 & \underline{\quad 37 \quad} & & \\
2 & \underline{\quad 18 \quad} & 1 & \uparrow \\
2 & \underline{\quad 9 \quad} & 0 & | \\
2 & \underline{\quad 4 \quad} & 1 & | \\
2 & \underline{\quad 2 \quad} & 0 & | \\
2 & \underline{\quad 1 \quad} & 0 & | \\
 & \underline{\quad 0 \quad} & 1 & |
\end{array}$$

$$(\mathbf{37})_{10} = (\mathbf{100101})_2$$

EXAMPLE 2.4 Convert 0.625 from Decimal to Binary system.

Solution :

$$\begin{array}{l l}
0.625 & \times 2 \\ \hline
1.250 & \times 2 \\ \hline
0.500 & \times 2 \\ \hline
1.000 &
\end{array}$$

$$(\mathbf{0.625})_{10} = (\mathbf{0.101})_2$$

EXAMPLE 2.5 Convert $(19.25)_{10}$ into Binary.

Solution :

$$\begin{array}{r|l l l}
2 & \underline{\quad 19 \quad} & & \\
2 & \underline{\quad 9 \quad} & 1 & | \\
2 & \underline{\quad 4 \quad} & 1 & | \\
2 & \underline{\quad 2 \quad} & 0 & | \\
2 & \underline{\quad 1 \quad} & 0 & | \\
 & \underline{\quad 0 \quad} & 1 & \downarrow
\end{array}
\qquad\qquad
\begin{array}{l|l l}
| & .25 & \times 2 \\ \cline{2-3}
| & 0.50 & \times 2 \\ \cline{2-3}
\downarrow & 1.00 &
\end{array}$$

$$(\mathbf{19})_{10} = (\mathbf{10011}) \qquad\qquad (\mathbf{.25})_{10} = (\mathbf{.01})_2$$

$$\therefore \quad (\mathbf{19.25})_{10} = (\mathbf{10011.01})_2$$

EXAMPLE 2.6 Convert $(142)_{10}$ into Hexadecimal system.

Solution :

$$\begin{array}{r|l l l}
16 & \underline{\quad 142 \quad} & & \\
16 & \underline{\quad 8 \quad} & E & \uparrow \\
 & \underline{\quad 0 \quad} & 8 & |
\end{array}$$

$$(\mathbf{142})_{10} = (\mathbf{8E})_{16}$$

It may be observed here that the decimal number 142 of 3 digits will be of only 2 digits in the Hexadecimal form. As such, large numbers can be stored in the available memory area if they are stored in the Hexadecimal form instead of in the Decimal form.

EXAMPLE 2.7 Convert $(1465.5)_{10}$ into Hexa form.

Solution :

$$\begin{array}{r|ll} 16 & \underline{\ 1465\ } & \\ 16 & \underline{\ 91\ } & 9 \\ 16 & \underline{\ 5\ } & B \\ & \underline{\ 0\ } & 5 \end{array}\uparrow \qquad\qquad \begin{array}{l} 0.5 \times 16 \\ \hline \downarrow\ 8.0 \end{array}$$

$\mathbf{(1465)_{10} = (5B9)_{16}}$ $\qquad$ $(0.5)_{10} = (0.8)_{16}$

$\therefore \mathbf{(1465.5)_{10} = (5B9.8)_{16}}$

EXAMPLE 2.8 Convert $(463.75)_{10}$ into Octal.

Solution :

$$\begin{array}{r|ll} 8 & \underline{\ 463\ } & \\ 8 & \underline{\ 57\ } & 7 \\ 8 & \underline{\ 7\ } & 1 \\ & \underline{\ 0\ } & 7 \end{array}\uparrow \qquad\qquad \begin{array}{l} 0.75 \times 8 \\ \hline \downarrow\ 6.00 \end{array}$$

$\mathbf{(463)_{10} = (717)_{8}}$ $\qquad$ $(0.75)_{10} = (0.6)_{8}$

$\therefore \mathbf{(463.75)_{10} = (717.6)_{8}}$

2.5.1 Rounding off Fraction

It will be found that the integer numbers in the Decimal system can be expressed by an equivalent number in the other systems. This may not be possible in many cases of fractional numbers.

For example, the conversion of (0.7) into Hexa will be as follows—

$$\begin{array}{ll} 0.7 & \times 16 \\ \hline 11.2 & \times 16 \\ \hline 3.2 & \times 16 \\ \hline 3.2 & \times 16 \\ \hline 3.2 & \end{array} \quad \text{and so on.}$$

In such cases, the stage at which the fractional portion becomes 0 will never be reached at all.

In such cases, calculations are done up to an extra digit, and half the value of the new base is added to it and the result is truncated to the desired number of digits. The following example will illustrate this.

EXAMPLE 2.9 Convert $(0.8)_{10}$ into Octal to 5 digit accuracy.

Solution :

$$\begin{array}{ll} 0.8 & \times 8 \\ \hline 6.4 & \times 8 \\ \hline 3.2 & \times 8 \\ \hline 1.6 & \times 8 \\ \hline 4.8 & \times 8 \\ \hline 6.4 & \times 8 \\ \hline 3.2 & \end{array} \qquad \mathbf{(0.8)_{10} = (.631463)_{8}}$$

This has been obtained to 6 digits. To get it to 5 digit accuracy adding 4 (half of new base 8) to the extra digit we get

$$.631463 + 4 = .631467.$$

Truncating it to 5 digits we get 0.63146.

So $(\mathbf{0.8})_{10} = (\mathbf{0.63146})_8$ to 5 digit accuracy.

EXAMPLE 2.10 Convert $(26.8)_{10}$ into Binary; the fractional part of the binary form should be of 4 digits.

Solution :

$$\begin{array}{r|l l} 2 & 26 & \\ 2 & 13 & 0 \\ 2 & 6 & 1 \\ 2 & 3 & 0 \\ 2 & 1 & 1 \\ & 0 & 1 \end{array}\Bigg\uparrow \qquad\qquad \Bigg\downarrow\begin{array}{l l} 0.8 & \times 2 \\ \hline 1.6 & \times 2 \\ \hline 1.2 & \times 2 \\ \hline 0.4 & \times 2 \\ \hline 0.8 & \times 2 \\ \hline 1.6 & \end{array}$$

$(\mathbf{26})_{10} = (\mathbf{11010})_2$ $\qquad$ $(0.8)_{10} = (\mathbf{0.11001})_2$

Adding 1, which is half the base, namely, 2, to the 5th digit, we get

$$\begin{array}{r} 0.11001 \\ \underline{1} \\ \underline{0.11010} \end{array}$$

Truncating this to 4 digits we get 0.1101

$$(0.8)_{10} = (0.1101)_2$$

So $(\mathbf{26.8})_{10} = (\mathbf{11010.1101})_2$.

2.6 CONVERSION OF A NUMBER FROM OTHER SYSTEMS TO THE DECIMAL SYSTEM

Let the number in any system with Base B (other than the Decimal System) have p integer digits and q fractional digits. Its Decimal value D will be given by—

$$D = C_{p-1}B^{p-1} + C_{p-2}B^{p-2} + \cdots + C_1 B + C_0 B^0 + C_{-1}B^{-1} + \cdots + C_{-q}B^{-q}$$

where $C_{p-1}, C_{p-2}, \cdots, C_{-q}$ are the coefficients in the respective digits. The following examples will illustrate the above.

2.6.1 Binary To Decimal

EXAMPLE 2.11 Convert $(10101.101)_2$ into Decimal System:

Solution :

$$\begin{aligned} D &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 16 + 0 + 4 + 0 + 1 + 0.5 + 0 + 0.125 = (21.625)_{10}. \end{aligned}$$

This can also be derived as follows—

$$
\begin{array}{lrcr}
1\ 0\ 1\ 0\ 1\ .\ 1\ 0\ 1 & & & \\
 & 1 \times 2^{-3} & = & 0.125 \\
 & 0 \times 2^{-2} & = & 0.000 \\
 & 1 \times 2^{-1} & = & 0.500 \\
 & 1 \times 2^{0} & = & 1.000 \\
 & 0 \times 2^{1} & = & 0.000 \\
 & 1 \times 2^{2} & = & 4.000 \\
 & 0 \times 2^{3} & = & 0.000 \\
 & 1 \times 2^{4} & = & \underline{16.000} \\
 & & & \underline{21.625}
\end{array}
$$

2.6.2 Octal to Decimal

EXAMPLE 2.12 Convert $(567.24)_8$ into Decimal form.

Solution :

$$
\begin{aligned}
D &= 5 \times 8^2 + 6 \times 8^1 + 7 \times 8^\circ + 2 \times 8^{-1} + 4 \times 8^{-2} \\
&= 320 + 48 + 7 + 0.25 + 0.0625 = (375.3125)_{10}.
\end{aligned}
$$

This can also be derived as follows—

$$
\begin{array}{lrcr}
5\ 6\ 7\ .\ 2\ 4 & & & \\
 & 4 \times 8^{-2} & = & 0.0625 \\
 & 2 \times 8^{-1} & = & 0.2500 \\
 & 7 \times 8^{\circ} & = & 7.0000 \\
 & 6 \times 8 & = & 48.0000 \\
 & 5 \times 8^{2} & = & \underline{320.0000} \\
 & & & \underline{375.3125}
\end{array}
$$

2.6.3 Hexadecimal to Decimal

EXAMPLE 2.13 Convert $(25A \cdot C8)_{16}$ into the Decimal equivalent.

Solution :

$$
\begin{aligned}
D &= 2 \times 16^2 + 5 \times 16 + A \times 16^\circ + C \times 16^{-1} + 8 \times 16^{-2} \\
&= 2 \times 16^2 + 5 \times 16 + 10 \times 16^\circ + 12 \times 16^{-1} + 8 \times 16^{-2} \\
&= 512 + 80 + 10 + 0.75 + 0.031 = (602.781)_{10}
\end{aligned}
$$

This can also be derived as follows—

$$
\begin{array}{lrcr}
2\ 5\ A\ .\ C\ 8 & & & \\
 & 8 \times 16^{-2} & = & 000.031 \\
 & 12 \times 16^{-1} & = & 000.750 \\
 & 10 \times 16^{\circ} & = & 10.000 \\
 & 5 \times 16 & = & 80.000 \\
 & 2 \times 16^{2} & = & \underline{512.000} \\
 & & & \underline{602.781}
\end{array}
$$

2.7 CONVERSION AMONG HEXADECIMAL, OCTAL AND BINARY SYSTEMS

2.7.1 Octal to Binary

An Octal number is converted into Binary form by replacing each octal digit by its equivalent THREE PLACE BINARY NUMBER. The following example will illustrate this.

EXAMPLE 2.14 Convert $(753.64)_8$ into Binary.

Solution :

7	5	3	.	6	4
111	101	011	.	110	100

$$\mathbf{(753.64)_8 = (111\ \ 101\ \ 011\ .\ 1101)_2}$$

2.7.2 Binary To Octal

To convert a Binary number into Octal form, group the binary digits into groups of 3 digits each, starting from the decimal point towards the left on the integer portion and towards the right on the fractional portion and replace each group by the equivalent Octal digit (refer Example 2.15).

EXAMPLE 2.15 Convert $(11101011.11)_2$ into Octal form.

Solution :

011	101	011	.	110
3	5	3	.	6

$$\mathbf{(011\ \ 101\ \ 011.11)_2 = (353.6)_8}.$$

2.7.3 Hexadecimal to Binary

To convert a Hexadecimal number into Binary, each digit in the Hexadecimal number is to be replaced by a four place Binary number. This is illustrated in the Example 2.16.

EXAMPLE 2.16 Convert $(9D7.A8)_{16}$ into Binary.

Solution :

9	D	7	.	A	8
1001	1101	0111	.	1010	1000

$$\mathbf{(9D7.A8)_{16} = (1001\ \ 1101\ \ 0111\ .\ 10101)_2}.$$

2.7.4 Binary to Hexadecimal

The method of converting a Binary number into Hexadecimal is somewhat similar to Binary to Octal conversion. In this case the Binary number is divided into groups of 4 digits each, starting from the decimal point towards the left in the integer portion and towards the right in the fractional portion and replacing each group by a Hexadecimal digit.

The following example illustrates the above method.

EXAMPLE 2.17 Convert (101 1100 1011 . 1101 01) into Hexadecimal.

Solution :

0101	1100	1011	.	1101	0100
5	C	B	.	D	8

$$\mathbf{(0101\ \ 1100\ \ 1011\ .\ 1101\ \ 01)_2 = (5CB.D8)_{16}}.$$

2.7.5 Octal to Hexadecimal

The Octal number is first expressed in the Binary form, from which the Hexadecimal form is derived as illustrated in the next example.

EXAMPLE 2.18 Convert $(576.67)_8$ into Hexadecimal.

Solution :

```
Octal Form           5    7    6   .   6    7
Equivalent Binary  101  111  110   .  110  111
```

Regroup the above Binary digits into groups of 4 digits each as follows—

```
                          0001 0111 1110 . 1101 1100
Equivalent Hexa number      1    7    E  .   D    C
```

$$(576.67)_8 = (17E.DC)_{16}.$$

2.7.6 Hexadecimal to Octal Conversion

The Hexadecimal number is expressed in the Binary form from which the Octal number is derived. The following example illustrates this.

EXAMPLE 2.19 Convert (AF5.C8) into Octal.

Solution :

```
Hexa Number          A    F    5   .   C    8
Equivalent Binary 1010 1111 0101   . 1100 1000
```

The above binary digits are regrouped into groups of 3 bits as follows—

```
             101  011  110  101  .  110  010
Octal form    5    3    6    5   .   6    2
```

$$(AF5.C8)_{16} = (5365.62)_8.$$

TABLE 2.1

Decimal Number	Binary Form	Octal Form	Hexadecimal Form
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

2.8 COMPUTER ARITHMETIC

It would be observed that, in the binary form, any number is represented by a group of 1's and 0's. This facilitates an easy design of the circuitry for computing which will be made up of electronic devices that can exist in two distinct physical states corresponding to TRUE, or 1 and FALSE, or 0. Hence, the arithmetic operations in a computer are done with the numeric data in the binary form.

Summary

In a computer system, numeric data are entered by the user in the Decimal form. The data is stored in the memory in the Hexadecimal form. The processor performs calculations with the data in the binary form. The result will also be stored in the memory in the Hexadecimal form. It will be displayed on the output device in the Decimal form.

Exercise/Questions

2.1 What are the two types of the number systems?

2.2 State the rules followed by the positional system.

2.3 Define the terms—(a) Radix, and (b) Weight, with respect to the number systems.

2.4 State and explain the formula used to evaluate the decimal value of a number in any other system.

2.5 In which form are the numeric data stored in the computer system and why?

2.6 On which form of the numeric data are the arithmetic operations performed in the computer and why?

2.7 Convert $(110101011111.011111)_2$ into—(a) Octal, (b) Decimal, and (c) Hexadecimal.

2.8 Convert $(7531.0246)_8$ into—(a) Binary, (b) Decimal, and (c) Hexadecimal.

2.9 Convert $(97531.20846)_{10}$ into—(a) Binary, (b) Octal, and (c) Hexadecimal.

2.10 Convert $(B9D8C7.E5F7)_{16}$ into—(a) Binary, (b) Octal, and (c) Decimal.

for NOTES

Chapter Three

Numeric Codes

3.1 DEFINITION

Representing numbers within the computer circuits, registers and the memory unit by means of Electrical signals or Magnetism is called NUMERIC CODING. As stated earlier, in the computer system, the numbers are stored in the Binary form, since any number can be represented by the use of 1's and 0's only. A binary 1 can be represented by the presence of a voltage or current pulse or magnetism and the binary 0, by the absence of it.

Some of the Memory units use eight tiny Magnetic rings of Ferrite material in each of their locations. A magnetised ring represents 1 and a demagnetised ring, 0. Each such ring is called a BIT, which is an abbreviation for BINARY DIGIT.

Instead of Ferrite rings, semiconductors can be used as Bits in the memory. If they conduct, they will represent the binary 1, else 0.

Some memory units use Electronic devices known as FLIP-FLOPS instead of semiconductors. The flip-flops have a terminal referred as Q. A charge in this terminal represents the binary 1 and absence of charge 0.

Instead of flip-flops, capacitors (condensers) can be used in the memory units. In such case a charged capacitor will represent a binary 1 and a discharged capacitor 0.

In the early stages of evolution of computers and memory units, several methods were tried to configure the numbers in them using the binary digits. These configurations are known as NUMERIC CODES.

3.2 CLASSIFICATION OF THE NUMERIC CODES

The Numeric Codes are broadly classified as—(i) Weighted Codes, and (ii) Non-weighted Codes.

In the Weighted Code, each bit in the number has a value depending upon its position or location within the number and this value is called the Weight. The Decimal value of the number will be equal to the sum of the product of the bits and their respective weights.

In the Non-weighted Codes the decimal number stored will not be equal to the bit configuration value.

3.3 WEIGHTED CODES

The different type of the Weighted Codes are—(i) BCD Code, (ii) $2-4-2-1$ Code, (iii) $4-2-2-1$ Code, (iv) $5-2-1-1$ Code, (v) $7-4-2-1$ Code, and (vi) $8-4-\bar{2}-\bar{1}$ Code.

3.3.1 BCD Code

BCD means BINARY CODED DECIMAL. In this code, a Decimal digit is expressed in the form of 4 BITS (Binary Digits). The least significant Bit has a Weight of 2^0, the next Bit to the left 2^1, the next to the left 2^2 and the most significant Bit 2^3 (refer Table 3.1). Some of the very early versions of the computers used this code for storing the digits 0 to 9. Later on it was found that using 4 Bits, numbers up to 15 (F in Hexadecimal) can be stored. So the practice of storing in the Hexadecimal form started. As stated in Chapter 2, larger numbers can be stored in the given locations if they are in Hexadecimal form than in the Decimal form.

TABLE 3.1

Decimal Number	BCD CODE			
	Bit–3 Weight $= 2^3 = 8$	Bit–2 Weight $= 2^2 = 4$	Bit–1 Weight $= 2^1 = 2$	Bit–0 Weight $= 2^0 = 1$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

3.3.2 Other Weighted Codes

The other types of Weighted Codes are given in the Table 3.2. Of these the $8\ 4\ \bar{2}\ \bar{1}$ code has two bits, namely $\bar{2}$ and $\bar{1}$ which have negative weights.

The above codes were some of the codes that were tested during the evolution of the digital computers. These are of very little significance now.

3.4 NON-WEIGHTED CODES

The Non-Weighted Codes are of two types, namely—(i) Non-Error Detecting Codes and, (ii) Error Detecting Codes.

TABLE 3.2

Decimal Number	Code 2421	Code 4221	Code 5221	Code 7421	Code $84\bar{2}\bar{1}$
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0111
2	0010	0010	0100	0010	0110
3	0011	0011	0101	0011	0101
4	0100	1000	0110	0100	0100
5	0101	1001	1000	0101	1011
6	1100	1010	1001	0110	1010
7	1101	1011	1100	1000	1001
8	1110	1110	1101	1001	1000
9	1111	1111	1110	1010	1111

3.4.1 Non-Error Detecting Codes

The Non-Error Detecting Codes are—(i) RING COUNTER Code, (ii) EXCESS THREE Code, and (iii) GRAY Code.

3.4.1.1 Ring Counter Code

This type of code was used in the first-ever Electronic Digital Computer, ENIAC. This uses totally TEN Bits to represent any digit 0 to 9 (refer Table 3.3). It may be noted that, though 10 Bits are used, only one Bit will be 'on' at a time.

TABLE 3.3

Decimal Number	Ring Counter Code
0	0000000001
1	0000000010
2	0000000100
3	0000001000
4	0000010000
5	0000100000
6	0001000000
7	0010000000
8	0100000000
9	1000000000

3.4.1.2 Excess Three Code

This is a Binary code in which each decimal number is expressed in excess of three, ie, 3 is added to the decimal number and then coded into Binary digits, eg, to get the Excess Three Code of 4, 3 is added to it which gives 7. The BCD code of 7, namely 0111 will be 4 in the Excess Three Code (refer Table 3.4). The Excess Three Code is said to be Self-Complementing code.

The complement of any single digit in any number system is the difference between the maximum value of a discrete character in that system and the number under consideration. (eg,) In the Decimal system, the complement of 6 is 3 (ie,) (9 – 6). So also the complement of 3 will be 6.

In a Binary Code, 1 and 0 are each complement to the other. The BCD code is not a self-complementing code in the sense that the Binary Form complement of a decimal number will not be its Decimal form complement. (eg,) 6 in BCD Code is 0110; complement of this Binary form is obtained by replacing all 1's by 0's and vice versa. In this case, it will be 1001, which is 9 in the Decimal system and not 3.

The decimal digit 6 will be expressed in the Excess Three code as 1001. Its Binary complement is 0110. This represents 3 in the Excess Three Code itself. This applies to all decimal digits 0 to 9 (refer Table 3.4). Because of this, Excess Three Code is said to be self-complementing.

TABLE 3.4

Decimal Number	Excess 3 Code	Complement of Excess 3 Code	Decimal value of the Complement
0	0011	1100	9
1	0100	1011	8
2	0101	1010	7
3	0110	1001	6
4	0111	1000	5
5	1000	0111	4
6	1001	0110	3
7	1010	0101	2
8	1011	0100	1
9	1100	0011	0

3.4.1.3 Gray Code

The other names for this code are—(i) UNIT DISTANCE CODE, (ii) CYCLIC CODE, and (iii) REFLECTED CODE.

It is called Unit Distance Code because the HAMMING DISTANCE between the Binary forms of any two adjacent numbers will always be one.

The Hamming Distance of any two numbers of equal length in the Binary form is defined as the number of positions by which they differ.

For example, the BCD code of 7 is 0111 and of 8 is 1000. The Hamming Distance between them is 4 since they differ in all the four digits. The same 7 in the Gray Code will be 0100 and 8 will be 1100 and the Hamming Distance between them in this code is 1 only. It will be seen from Table 3.5 that the Hamming Distance of any consecutive two numbers (0 to 9) in the Gray Code is always 1.

TABLE 3.5

Decimal Number	Gray Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1000

Gray code of a Decimal can be obtained from its BCD code as follows—

(i) Write down its BCD Code form;

(ii) The most significant Bit in the Gray Code is the same as in the BCD Code;

(iii) The second and the successive bits in the Gray Code will be the sum of the Bit in the same position and the previous position in the BCD Code.

While calculating the sum, the following guidelines should be observed—

(i) $0 + 0 = 0$;

(ii) $0 + 1 = 1$;

(iii) $1 + 1 = 0$ (with a carry of 1, which is to be neglected).

EXAMPLE 3.1 Find the Gray Code of the Decimal number 5.

Solution :

5 in BCD Code 0 1 0 1
↓
5 in Gray Code 0 1 1 1

Please note this method is not applicable to the Decimal number 9. Its Gray Code will be 1000 and not 1101. This is adopted to maintain a Hamming Distance of 1 between 9 and 0.

In Electronic circuits, devices known as XOR Gates are used for the conversion of BCD Code into Gray Code.

To convert a number from the Gray Code to the BCD Code the following steps are to be used:

(i) The most significant Bit in the BCD Code is the same as that in the Gray Code.

(ii) The second and the succeeding bits will each be the sum of the corresponding bit in the Gray Code and the previous bit in the BCD Code. The rules of addition are the same as for the conversion of BCD to Gray Code.

EXAMPLE 3.2 Find the Decimal value of the Gray Code Number 1100.

Solution :

Gray Code Number 1 1 0 0
Equivalent BCD Code 1 0 0 0

The Decimal number = **8.**

Gray Code is useful in converting varying analog signals into Digital form since only one bi varies from one Code to the next.

3.4.2 Error Detecting Codes

These are of 2 types, namely—(i) ERROR CHECKING CODES, and (ii) ERROR CORRECTIN CODES.

3.4.2.1 Error Checking Codes

These are of 2 categories, namely—(i) SELF-CHECKING CODES, and (ii) PARITY CHECKIN CODES.

3.4.2.1.1 Self-Checking Codes

In this type of Code, though the configuration of the Bits will be different for different digits (0 to 9), the number of 1 Bits will be the same for all the digits. The BIQUINARY CODE and the '2 out of 5 Code' belong to this category. In them, the total number of 1 Bits in the configuration of any number is always 2 (refer Table 3.6). Because of this, any error due to the failure or absence of any Bit can easily be recognised.

The Biquinary Code was adopted in the Japanese ABACUS which was called SOROBAN. But it is not in use now since it has a bit of weight 5. So also the '2 out of 5 code' is not in use since it has a bit of weight 7.

TABLE 3.6

Decimal Number	**Biquinary Code** **5043210**	**2 out of 5 Code** **74210**
0	0100001	11000
1	0100010	00011
2	0100100	00101
3	0101000	00110
4	0110000	01001
5	1000001	01010
6	1000010	01100
7	1000100	10001
8	1001000	10010
9	1010000	10100

3.4.2.1.2 Parity Checking Codes

There are 2 types of PARITY CHECKING Codes, namely—(i) ODD PARITY CODE, and (ii) EVEN PARITY CODE.

In both these codes, in addition to the Numeric Bits, an additional bit is used. This bit is called the PARITY BIT or the CHECK BIT.

The coding system can be designed either for Odd Parity or for Even Parity. If designed for the Odd Parity, the parity bit will be automatically generated in case the number of 1 bits in the numeric code is even. Similarly, if designed for Even Parity, the Parity bit will be automatically generated if the number of 1's in the numeric bits is odd (refer Table 3.7). Most of the computers are designed for Odd Parity.

In the Table 3.7, D_0, D_1, D_2 and D_3 stand for the numeric bits of weights $2^0, 2^1, 2^2$ and 2^3 respectively. P_o denotes Odd Parity bit and P_e Even Parity bit. It may be noted that zero is represented as 8 2 in the Even bit configuration.

3.4.2.2 Error Checking Codes

The following types of codes come under this classification—(i) Simple Error Correcting Code, (ii) Self Correcting Code, and (iii) Hamming Code.

3.4.2.2.1 Simple Error Correcting Code

This is adopted for recording Data in the Magnetic Tapes. Data is recorded in them in 7 or 9 tracks. In a 9 track system, characters are recorded by energised Magnetic Spots on 8 tracks. The

TABLE 3.7

Decimal Number	(a) Odd Parity Configuration $D_3D_2D_1D_0P_e$	(b) Even Parity Configuration $D_3D_2D_1D_0P_e$
0	00001	10100
1	00010	00011
2	00100	00101
3	00111	00110
4	01000	01001
5	01011	01010
6	01101	01100
7	01110	01111
8	10000	10001
9	10011	10010

configuration of these spots will be different for different characters. The 9th track functions as the Parity track. In a system designed for Odd Parity, the 9th track will be energised, if necessary, in a column to maintain Odd Parity. Besides this, for each element of Data across each track a CHECK BIT is designed to be generated to maintain Odd Parity along each track also. Table 3.8 gives the way the Decimal number 5384 will be recorded on Tape in the EBCDIC code and how the Parity and the Check bits will be generated along the column and track/row, respectively.

TABLE 3.8

Tracks	DATA 5384	Track Check-Bit
D	1100	1
D	0100	0
D	1001	1
D	0010	0
D	1111	1
D	1111	1
D	1111	1
D	1111	1
P	1100	1

3.4.2.2.2 Self-Correcting Code

This code has also application in the case of recording data on Magnetic tapes. Each character is recorded odd number of times (3 or 5 times) in adjacent columns. In case the character in any one of the columns is not intelligible—due to the scratches or formation of dust on the tape-surface OR it is different from the character in the adjacent columns—the computer is designed to accept the data in the majority of the columns. For example, Table 3.9 shows recording of Character 7 in ASCII Code in successive 3 columns instead of only once. If the recording at any spot is blurred (spot marked as X) the processor will read the data from the majority configuration.

3.4.2.2.3 Hamming Code

Hamming Codes are named after R W Hamming who formulated the method of Multiple

TABLE 3.9

1	1	1
1	X	1
1	1	1
0	0	0
1	1	1
1	1	1
0	0	0
0	0	0

Parity Checks. Let $P_0, P_1 \cdots$ etc. $\cdots P_n$ be the parity bits used in the code. These will be scattered through the message (ie,) among the weighted bits, say, $D_0, D_1 \cdots$ etc. $\cdots D_m$ representing the character. The Desired Odd or Even Parity configuration will be designed to be maintained for different combinations. Example 3.3 illustrates a method of formulating a type of Hamming Code.

EXAMPLE 3.3 Three Parity bits are to be designed for error checking in the BCD coded decimal numbers. These bits are to be distributed among the weighted bits. If the weighted bits are denoted as D_0, D_1, D_2 and D_3 and the Parity bits P_0, P_1 and P_2, the Hamming Code is to be designed for ODD Parity for each one of the following combinations—(i) P_0, P_1, P_2 and D_2, (ii) P_1, D_0, D_2 and D_3, and (iii) P_1, D_1, D_2 and D_3.

Solution : Since the Parity Bits are to be distributed among the weighted bits, the configuration will be $D_3, P_2, D_2, P_1, D_1, P_0$ and D_0.

The way the Decimal numbers are represented by the above bits, maintaining the Odd Parity for the given combinations, is given in Table 3.10.

TABLE 3.10

Decimal Number	D_3	P_2	D_2	P_1	D_1	P_0	D_0
0	0	1	0	1	0	1	0
1	0	1	0	0	0	0	1
2	0	0	0	1	1	0	0
3	0	0	0	0	1	1	1
4	0	0	1	0	0	0	0
5	0	0	1	1	0	1	1
6	0	1	1	0	1	1	0
7	0	1	1	1	1	0	1
8	1	0	0	0	0	1	0
9	1	0	0	1	0	0	1

Note: Wrong parity in any one or more combinations will indicate an Error Condition.

3.5 ALPHAMERIC CODES

Though the Digital Computer was primarily designed to process Numeric Data, it has been now improvised to process STRING Data which includes, besides the Digits, the alphabets and signs like $+, -, =$ etc. So codes have been developed for recording the various Characters on tapes and disks and also for storing them in the Memory of the Computer. The code used for this purpose by the IBM Computers is known as EBCDIC (Extended Binary Coded Decimal Information/Interchange Code) Code. The other Computers adopt the ASCII (American Standard Code for Information Interchange) Code. The ASCII and the EBCDIC forms of the various characters are given in Table 3.11.

TABLE 3.11

Character	ASCII Code	EBCDIC Code
0	00110000	11110000
1	00110001	11110001
2	00110010	11110010
3	00110011	11110011
4	00110100	11110100
5	00110101	11110101
6	00110110	11110110
7	01100111	11110111
8	01101000	11111000
9	01101001	11111001
A	01000001	11000001
B	01000010	11000010
C	01000011	11000011
D	01000100	11000011
E	01000101	11000100
F	01000110	11000110
G	01000111	11000111
H	01001000	11001000
I	01001001	11001001
J	01001010	11010001
K	01001011	11010010
L	01001100	11010011
M	01001101	11010100
N	01001110	11010101
O	01001111	11010110
P	01010000	11010111
Q	01010001	11011000
R	01010010	11011001
S	01010011	11100010
T	01010100	11100011
U	01010101	11100100
V	01010110	11100101
W	01010111	11100110
X	01011000	11100111
Y	01011001	11101000
Z	01011010	11101001
Decimal Point	00101111	01101011
Blank	00100000	01000000
+	00101011	01001110
–	00101101	01100000
*	00101010	01011100
/	00101100	01100001
(	00101000	01001101
)	00101001	01011101
=	00111101	01111110

Both the ASCII and the EBCDIC codes use 8 bit configuration for any character. The ASCII code provides for the lower case letters also (refer Table 3.12).

TABLE 3.12

Lower Case Letter	ASCII Code
a	01100001
b	01100010
c	01100011
d	01100100
e	01100101
f	01100110
g	01100111
h	01101000
i	01101001
j	01101010
k	01101011
l	01101100
m	01101101
n	01101110
o	01101111
p	01110000
q	01110001
r	01110010
s	01100011
t	01110100
u	01110101
v	01110110
w	01110111
x	01111000
y	01111001
z	01111010

Table 3.13 gives the ASCII Codes of certain symbols used in texts as also in computer-programs.

3.6 DISPLAY CODES

These Codes are used to display Data or result on output devices like the Printer, Screen and LED (Light Emitting Diode) Display Units. The display may be—(i) a Seven Segmental Display or (ii) Dot-Matrix Display.

3.6.1 Seven Segment Display

Fig 3.1 shows a Seven-Segment LED. Each segment is labelled by letter A to G. Different combinations of the seven segments are illuminated to display specific digits. For example, to display the digit 3, segments A, B, C, D and G will be illuminated. The Seven Segment LED display units are used in the Micro-Processor Kits to display Data and the results.

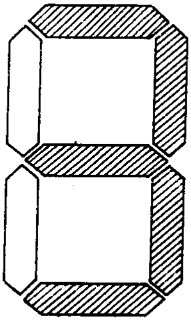


Fig 3.1

3.6.2 Dot-Matrix Display

This type of display is used on the output devices like the Screen and the Dot-Matrix Printer.

TABLE 3.13

Character	ASCII Code	Remarks
!	00100001	Exclamation Mark
"	00100010	Double Quotes
#	00100011	Hash Symbol
$	00100100	Dollar Sign
%	00100101	Percent
&	00100110	Ampersand
'	00100111	Single Quotes
,	00101100	Coma
:	00111010	Colon
;	00111011	Semicolon
<	00111100	Less Than
>	00111110	Greater Than
?	00111111	Question Mark
@	01000000	at the rate of
[	01011011	Left square Bracket
\	01011100	Backslash
]	01011101	Right Square Bracket
↑	01011110	Vertical Arrow

The output characters that are of 8-bit code are converted into a suitable Dot-Matrix pattern either to be displayed or printed. Fig. 3.2(a) illustrates a 5 Column by 7 Row Dot-Matrix configuration and 3.2(b) the way the letter 'E' will be represented in the dot-matrix form.

```
0 0 0 0 0          0 0 0 0 0
0 0 0 0 0          0
0 0 0 0 0          0
0 0 0 0 0          0 0 0 0
0 0 0 0 0          0
0 0 0 0 0          0
0 0 0 0 0          0 0 0 0 0
   (a)                (b)
```

Fig 3.2

Summary

Codes are used to configure the characters, in the computer system. The earlier IBM Computers used the BCD Code. The present-day IBM machines use the EBCDIC Code, and the other computers the ASCII Code. For the display of the output, the Seven-Segment Display or the Dot-Matrix display code is used.

The Excess 3 Code is sometimes used for storing the Digits since it is self-complementing. The Gray Code is Cyclic and hence used to store the analog data in the digital form.

Exercise/Questions

3.1 Explain why the characters are configured in the Binary Form in the computer system.

3.2 Differentiate between Weighted Codes and the Non-Weighted Codes.

3.3 Give the expansions of the terms—(i) BCD, (ii) EBCDIC, (iii) ASCII, and (iv) Bit.

3.4 Which of the Weighted Codes has bits of negative weights?

3.5 Show that the Excess 3 Code is self-complementing.

3.6 Define the term HAMMING DISTANCE.

3.7 State why the Gray Code is called a Cyclic Code? State its application.

3.8 State the rules for conversion—(i) From BCD to Gray Code, and (ii) Gray Code to BCD Code.

3.9 Explain the terms—(i) Parity Bit, (ii) Hamming Code, and (iii) Ring Counter Code.

3.10 Write short notes on—(i) Biquinary Code, (ii) Self-checking Code, and (iii) Parity Checking Code.

3.11 Write down the ASCII and the EBCDIC Codes for the following characters—(i) F, (ii) 6, and (iii) *.

3.12 Write down the characters represented by the following ASCII Codes—(i) 00110011, (ii) 01000100, (iii) 00111101, and (iv) 01110111.

3.13 Write down the characters represented by the following EBCDIC Codes—(i) 01101011, (ii) 11010100, and (iii) 11110111.

3.14 Describe the Seven Segment and Dot-Matrix Display Codes.

for NOTES

Chapter Four

Arithmetic Unit

4.1 INTRODUCTION

As stated in Section 1.3, the CPU contains an Arithmetic Logic Unit (ALU), a few General Purpose Registers, a Clock and a Control Unit. It has also a Status Register which is also called Flag Register or Conditional code Register. The ALU performs the Arithmetic and Logic operations. The Status Register is made use of in executing the logical operations. The Clock is an Electronic device which gives out Pulses of very high frequency (16 MHZ). The General Purpose Registers are

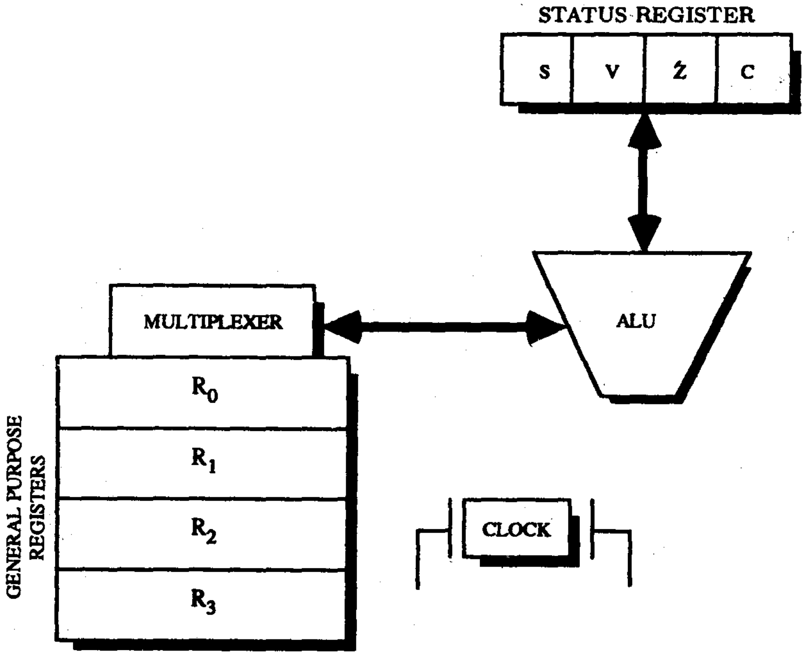

STATUS REGISTER
S
V
Z
C
ALU
MULTIPLEXER
R_0
R_1
R_2
R_3
GENERAL PURPOSE REGISTERS
CLOCK

Fig 4.1

used for—

(i) For storing the Data which is currently processed by the ALU;
(ii) For storing some data temporarily; and also
(iii) For temporarily storing the intermediate results.

The clock pulses are used to synchronise the various operations.

The function of the Control Unit is discussed in detail in the next chapter.

4.2 ARITHMETIC OPERATIONS

The computer is designed to perform the following Arithmetic Operations—(i) Addition, (ii) Subtraction, (iii) Multiplication, (iv) Division, and (v) Exponentiation. It performs these operations with the numeric data in the Binary form. In this form, any number is represented by a group of 1's and 0's. In the computer system, the binary 1 is represented by the presence of a current pulse or voltage and the binary 0, by its absence.

4.2.1 Binary Addition

In Binary Arithmetic $0+0=0; 0+1=1; 1+0=1; 1+1=0$ with a carry of 1.

This can be given in the form of a table which is called as TRUTH TABLE (refer Table 4.1).

Adding 2 Binary digits without considering the carry from the addition of the adjacent less significant digits is called Half-Addition.

TABLE 4.1 TRUTH TABLE FOR HALF-ADDITION

G	E	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

G: Augend; E: Addend; S: Sum; C: Resulting Carry.

The Electronic device that performs the Half-Addition is called as Half-Adder. The block diagram of a Half-Adder is shown in Fig. 4.2.

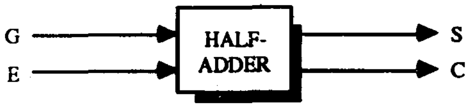


Fig 4.2

4.2.2 Full Addition

In the computers, the Augend and the Addend are mostly stored in separate General Purpose Registers and the ALU performs their addition. The result will be stored in any one of the 2 registers. In small computers, each Register will be of 8 Bits and addition of two 8 bit numbers can be performed in them. Medium size computers have 16 bit registers, Big size computers 32 bit registers and Supercomputers 64 bit Registers. When two 8 bit or 16 bit or 32 bit or 64 bit Binary numbers are added, the addition of the least significant bits will be an Half-Addition. In the case of the subsequent digits, the carry from the previous addition has also to be added along with the two digits. This addition is called FULL-ADDITION. The Truth-Table for full addition is shown in Table 4.2.

TABLE 4.2 TRUTH TABLE FOR FULL-ADDITION

G	E	P	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

G: Augend; E: Addend; P: Previous Carry; S: Sum; C: Resulting Carry.

The Electronic device that performs full-addition is called FULL-ADDER. The Block-Diagram of a Full-Adder is shown in Fig. 4.3.

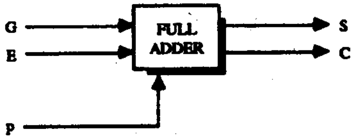


Fig 4.3 Full-Adder

The addition of 2 Binary Numbers with two or more digits is illustrated in the example below.

EXAMPLE 4.1 Perform the addition of the Binary numbers 1011 and 1010.

		Decimal Value
Augend	1011	11
Addend	1010	10
Sum	10101	21

4.2.3 Adder Circuits

One type of Adder Circuit will contain as many Adders as the number of bits in the (size of the) General Purpose Registers. One of them will be a Half-Adder and the rest Full-Adders. The Half-Adder is to add the least significant digits and the others for adding the other digits. Such an Adder is called a RIPPLE CARRY ADDER or PARALLEL ADDER. The block diagram of a 4 Bit Ripple Carry Adder is given in Fig. 4.4. Note that the Carry from each stage is taken to the next stage for adding and the 'Final Carry' is taken as the most significant bit in the Sum.

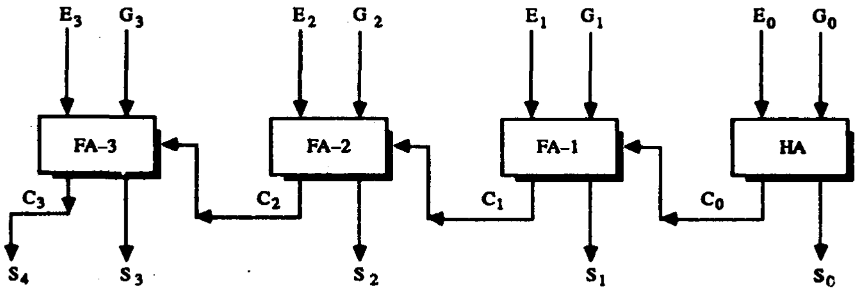


Fig 4.4 Ripple Carry Adder

HA: Half-Adder; FA-1, FA-2, FA-3: Full-Adders; G_0, G_1, G_2, G_3: Augend Bits; E_0, E_1, E_2, E_3: Addend Bits; C_0, C_1, C_2, C_3: Carry Bits; S_0, S_1, S_2, S_3, S_4: Sum Bits.

4.2.4 Serial Adder Circuit

This consists of only one Adder which will be a Full-Adder. The Augend and the Addend are each stored in a separate Register. The Binary Data in these Registers are simultaneously shifted Bit by Bit to the Right by a clock pulse. The shifted bits are added by the Full-Adder and the Sum bit is transferred to a Register. The circuit includes a CARRY FLIP-FLOP and a CYCLE COUNTER. The carry Flip-Flop takes out the Carry from the Adder and feeds back to it so that it will be added with the bits of the next higher order from the Augend and the Addend. The Carry Flip-Flop is initialised (made to 0) initially (ie) while adding the least significant bits. The Cycle Counter will be initialised at the start of the operation. After each clock-pulse (ie) after each addition, it will be incremented by 1. After the count in it becomes equal to the number of bits in any one of the registers, the Add Operation will be terminated.

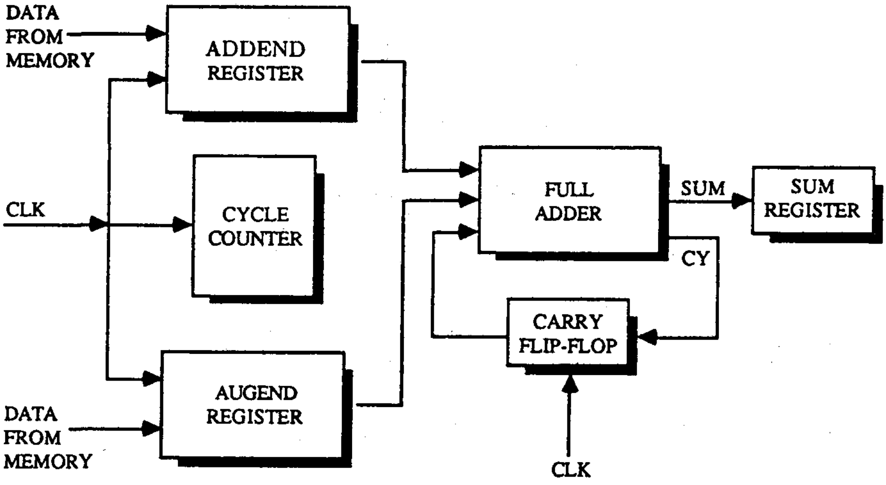


Fig 4.5 Serial Adder

4.3 BINARY SUBTRACTION

There are 2 methods of performing subtraction of a Binary number from another Binary number, namely—(i) Direct Subtraction, and (ii) Subtraction by COMPLEMENTARY Addition.

4.3.1 Direct Subtraction

In this: $1 - 1 = 0; 1 - 0 = 1; 0 - 0 = 0; 0 - 1 = 1$ with a borrow of 1. This way of subtracting a Binary digit from another without considering the previous borrow, if any, is called HALF SUBTRACTION. The Truth Table for Half Subtraction is given in the Table 4.3.

TABLE 4.3 TRUTH TABLE FOR HALF-SUBTRACTION

M	T	D = M-T	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

M: Minuend; T: Subtrahend; D: Difference; B: Resulting Borrow.

The Electronic device that performs Half-Subtraction is called Half-Subtractor. The Block diagram of a Half-Subtractor is given in Fig. 4.6.

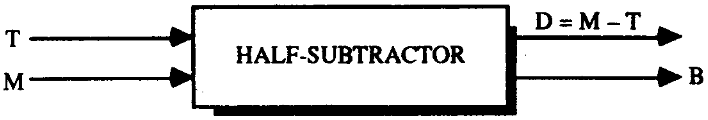


Fig 4.6 Half-Subtractor

4.3.1.1 Full Subtraction

If the Minuend and the Subtrahend are of two or more digits, the subtraction of the least significant digit of the subtrahend from that of the minuend is half-subtraction. For the subtraction of the higher order digits the borrow resulting from the previous operation has to be taken into account. Such a subtraction is called Full-Subtraction. The Truth Table for the Full-Subtraction is given in Table 4.4.

TABLE 4.4 TRUTH TABLE FOR FULL-SUBTRACTION

M	T	W	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

M: Minuend; T: Subtrahend; W: Previous Borrow; D: Difference; B: Resulting Borrow.

The Electronic Device that is designed to perform Full-Subtraction is called FULL-SUBTRACTOR (refer block diagram in Fig. 4.7).

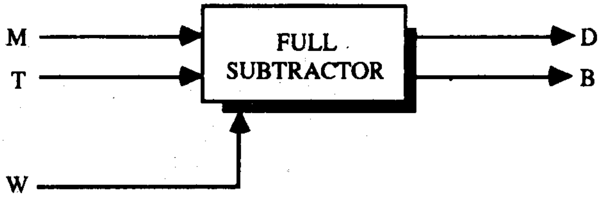


Fig 4.7 Full-Subtractor

The Subtraction of a Binary number of more than 2 digits from a similar number is illustrated in the Example 4.2.

EXAMPLE 4.2 Subtract 1 0 1 0 from 1 1 1 1:

Solution :

		Decimal Value
Minuend	1111	15
Subtrahend	1010	10
Difference	0101	05

In the present-day computers, the method of Direct subtraction is not adopted; the method of subtraction by complementing is used instead.

4.3.2 Subtraction by Complementing

The principle of subtraction by complementing can be illustrated as in Example 4.3.

EXAMPLE 4.3 Subtract $(46)_{10}$ from $(87)_{10}$ by complementing method.

Solution:

Minuend = 87.

Subtrahend = 46.

Complement of the Subtrahend = (99 − 46) = 53.

Add the minuend and the complement of the subtrahend $87 + 53 = \overset{\text{EAC}}{\overset{\downarrow}{1}}40.$

It is found that the result has an extra digit (ie) while the Minuend and the Subtrahend are of 2 digits, the results is of 3 digits. The extra digit (ie) the most significant digit in the result is called END AROUND CARRY (EAC).

This is removed and added to the least significant bit in the result itself to get the actual difference.

$$\begin{array}{r} 1\,4\,0 \\ \searrow 1 \\ \hline \underline{4\,1} \end{array}$$

In the above case, the magnitude of the Subtrahend is less than that of the Minuend. Example 4.4 describes a case where the magnitude of the Subtrahend is greater than that of the Minuend.

EXAMPLE 4.4 Subtract (75) from (54) by method of complementing.

Solution :

Minuend = 54.

Subtrahend = 75.

Complement of Subtrahend = (99 − 75) = 24.

Adding minuend and complement of the subtrahend the result is 54 + 24 = 78.

There is no EAC in the result. So the actual difference will be got by complementing the result as follows and prefixing a negative sign to it.

Complement of the result (99 − 78) = 21.

Prefixing a − sign, the difference = −21.

The same procedure as the above is followed when subtraction by complementing is done with the Minuend and the Subtrahend in the Binary form except that they are stored in The SIGN-Magnitude form.

4.3.3 Sign Magnitude Representation of Numbers

While storing numbers in the Memory and in the General Purpose Registers, to indicate whether it is a positive or negative number, the Most Significant Bit of the number is used as the sign bit. If this bit is 1, the number is negative; else (ie) if it is 0, it is positive. For example, if the contents of an Eight-Bit Register is: 1 0 1 0 0 0 0 1 the number stored is −33. If it is: 0 0 1 0 0 0 0 1, the number is 33.

4.3.4 Different Methods of Complementing

There are 3 methods of complementing a Binary number, namely—(i) 1's Complementing; (ii) 2's Complementing, and (iii) Characteristic method.

The 1's Complement of a Binary number is obtained by replacing the 1's in it by 0's and 0's by 1's.

EXAMPLE 4.5 Find the 1's complement of 1 0 1 0 1 0 0 1.

Solution :

Given Binary number	1 0 1 0 1 0 0 1
Its 1's Complement	0 1 0 1 0 1 1 0

4.3.5 Binary Subtraction by 1's Complementing

The procedure for subtraction of a Binary number from another Binary number will be as follows:

(i) Store both the Minuend and the Subtrahend in the Sign-Magnitude form.

(ii) Get the 1's complement of the Subtrahend, including its sign bit.

(iii) Add the 1's complement of the Subtrahend with the Minuend.

(iv) Check the result to find whether there is an EAC in it. If there is, remove it and add it to the Least Significant Digit of the result, to give the actual difference.

(v) If there is no EAC, 1's complement the result, to get the actual difference.

Examples 4.6 and 4.7 are illustrative of the above procedures.

EXAMPLE 4.6 Show how 5 is subtracted from 7, in Binary form, by 1's complementing method.

Solution :

Decimal Form	Binary Form	
	Sign Bit	Numeric Bits
7	0	111
−5	1	101
1's complement of (−5)	0	010
Adding 7 and the	0111	
1's complement of −5	0010	
the result is	1001	
Adding the EAC to the LSB	↘1	
Difference	0010	= $(2)_{10}$

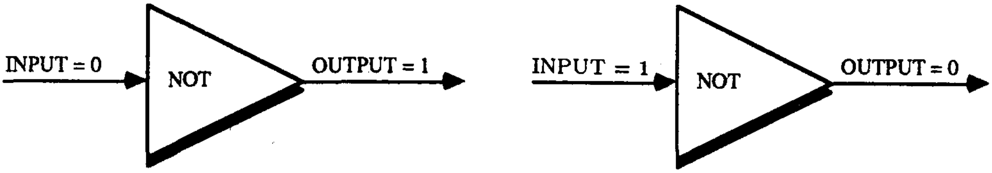
EXAMPLE 4.7 Show how 7 is subtracted from 5 in the Binary form using 1's complement method.

Solution :

Decimal Form	Binary Form	
	Sign Bit	Numeric Bits
5	0	101
−7	1	111
1's complement of (−7)	0	000
Adding 5 and the		
1's complement of −7	0	101
the result is	0	000
There is no EAC in the result	0	101
Complementing the result		
Difference	1	010=$(-2)_{10}$

4.3.6 Subtraction Circuit

For complementing a Binary digit a device called INVERTOR or NOT-Gate is used [refer Fig. 4.8 (a) and (b)].


INPUT = 0
NOT
OUTPUT = 1
INPUT = 1
NOT
OUTPUT = 0

(a) (b)

Fig 4.8 Invertor

The circuit for Subtraction by 1's complement method will have as many Full-Adders as the number of bits in the register including the sign bit. A Full-Adder is used even for the least significant bits since EAC, if any, has to be added with them. Invertors are used one for each digit to complement the digits of the subtrahend. The complemented digits of the subtrahend are added with the corresponding bits of the Minuend by the Full-Adders (refer Fig. 4.9).

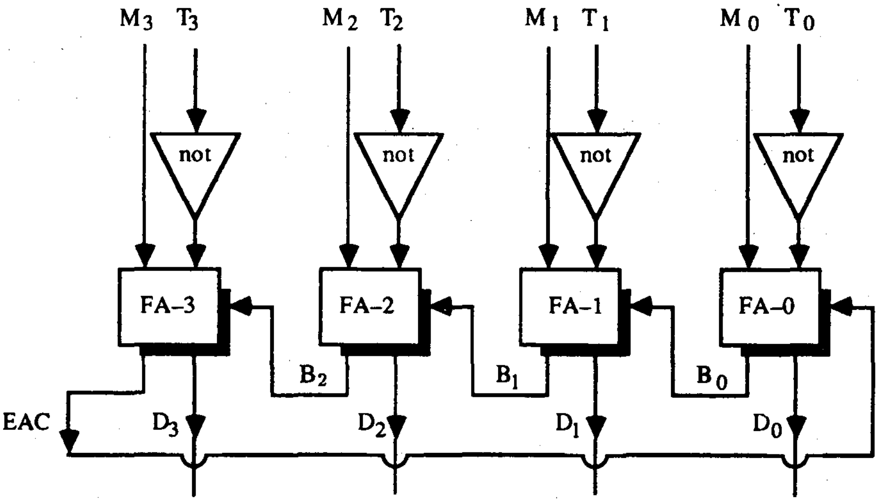

M_3 T_3
M_2 T_2
M_1 T_1
M_0 T_0
not
not
not
not
FA-3
FA-2
FA-1
FA-0
B_2
B_1
B_0
EAC
D_3
D_2
D_1
D_0

Fig 4.9 Binary Subtractor

4.3.7 Subtraction by 2's Complementing

The 2's complement of a Binary number is obtained as follows—

(i) Write its 1's complement; and

(ii) Add 1 to its Least Significant Bit.

This is illustrated in the example 4.8.

EXAMPLE 4.8 Write down the 2's complement of the Binary number 1010.

Solution :

Given Binary number	1010	→ ···
Its 1's complement	0101	→ ···
Add 1 to the LSB of the 1's complement	1	→ ···
2's complement	0110	→ ···

4.3.8 Alternate Method for 2's Complementing

There is another method of getting the 2's complement of a Binary number. The method is: "Scan the Binary number from right to left and complement all the bits appearing after the first '1' bit".

Example 4.9 illustrates this method.

EXAMPLE 4.9 Give the 2's complement of the Binary number 10 101010.

Solution :

Scanning the given number from right to left the first 1 bit that comes is next to the Least Significant Bit. Write down this 1 and the 0's to the right of it as they are and complement all the bits to its left:

The result will be 01010110.

4.3.9 Rules for Subtraction by 2's Complementing

The steps for Subtraction of a Binary number from another Binary number by 2's complementing method are—

(i) 2's complement the subtrahend.

(ii) Add the 2's complement of the Subtrahend with the Minuend.

(iii) Check the Sign Bit in the result.

(iv) If it is 0, the result will directly give the difference. (The magnitude of the Minuend will be greater than that of the Subtrahend in this case)

(v) Else, if the Sign Bit is 1, 2's complement the result to get the actual diffierence.

Note : The Sign Bit is not to be complemented in 2's complementing.

The above steps are illustrated in the Examples 4.10 and 4.11.

EXAMPLE 4.10 Assuming a 6 Bit configuration, inclusive of Sign Bit, show how 14 is subtracted from 23 by 2's complementing method.

Solution :

Decimal Form	Binary Form	
	Sign Bit	Numeric Bits
23	0	10111
−14	1	01110
2's complement of −14	1	10010
Adding 23 and 2's complement of −14	0 1	10111 10010
The Result is =	0	01001
(The sign bit is not present in the result; Hence the result itself is the difference)		
Difference ... ... =	0	01001 = 9

EXAMPLE 4.11 Assuming a 6 Bit configuration, inclusive of the Sign Bit, show how 23 is subtracted from 15, by 2's complementing method.

Solution :

Decimal Form	Binary Form	
	Sign Bit	Numeric Bits
15	0	01111
−23	1	10111
2's complement of −23	1	01001
Adding 15 and 2's complemnt of −23	0 1	01111 01001
The Result is =	1	11000
Since Sign Bit is present in the result 2's complement is =	1	01000 = −9

4.3.10 Combined Adder Subtractor Unit

In most of the present-day computers, the same unit is used to perform Addition and Subtraction. The subtraction will be by 2's complement method. Such a unit is called Combined Adder-Subtractor unit. (refer Fig. 4.10)

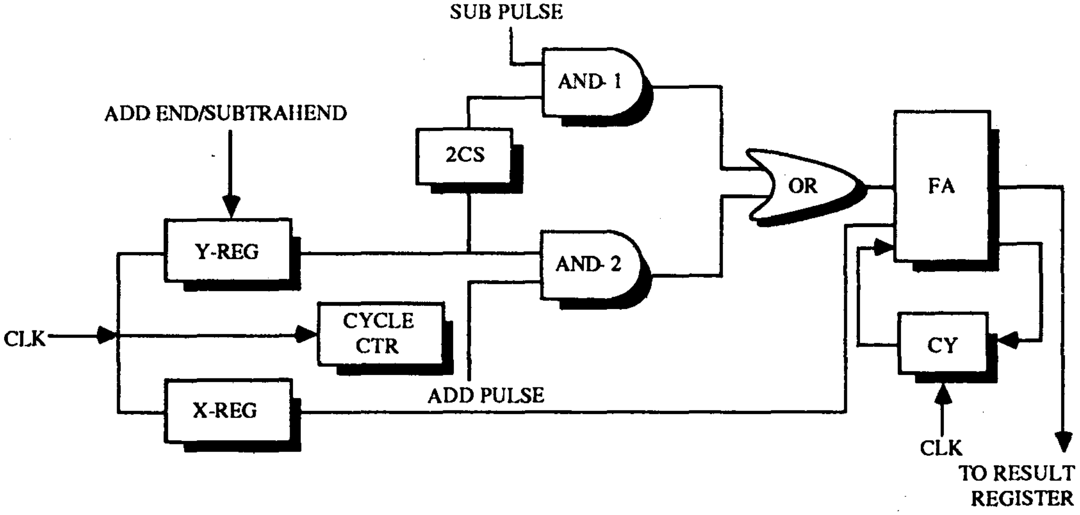


Fig 4.10 Combined Adder/Subtractor

The circuit contains 2 AND-GATES and an OR-GATE. An And-Gate is an Electronic device that can give an output only if there are minimum 2 inputs. An Or-Gate is an Electronic device which can give an output if there is even one input only.

The Augend or the Minuend is stored in the register marked as X-Reg. The Addend or the Subtrahend is stored in the other register called as Y-Reg. If the operation is addition, an Add-Pulse is generated. On a clock pulse, the bits in the 2 registers will be shifted one by one to the right. The bits from the X-Register will be transferred to the Full-Adder. The bits from the Y-Register will go through the And-Gate-2 and the Or-Gate to the Full-Adder. The Result will go to the Result-Register.

In case the operation is a subtraction, there will be a Subtract-Pulse, in which case the bits from Y-Register will be 2's complemented and will be transferred through the And-Gate-1 and the Or-Gate to the Full-Adder. In both cases, the result will be available in the Result Register. The Carry flip-flop will function just as in a serial Adder circuit.

4.3.11 Subtraction by the Characteristic Method

This method is used in the computers that use a 6 Bit sign-magnitude method of storing the Integer numbers. In this method, both the Minuend and the Subtrahend are expressed in the CHARACTERISTIC form. The Characteristic form of a Binary Number is its Binary Expression in excess of 32. For example, the Characteristic form of 17 is: 110001 which is 49(17 + 32). Similarly, −15 will be: 010001 which is 17(−15 + 32), in the Characteristic form.

In this method of subtraction, there is no need to find which is the subtrahend or the negative number, since both the numbers are to be expressed in the Characteristic form. It will be found that, for a positive number, the magnitude bits are the same both in the Binary and the Characteristic forms. For the negative number, the magnitude bits in the Characteristic form will be the 2's complement of the magnitude bits in the Binary form.

In this method, the steps are: (i) Express both the Minuend and the Subtrahend in the Characteristic form. (ii) Add both of them in their Characteristic form. (iii) Check whether the sign bit is present in the result. If it is not present, the result itself gives the difference. (iv) If the Sign bit is present 2's complement the result to get the actual difference.

Example 4.12 and 4.13 will illustrate these steps.

EXAMPLE 4.12 Illustrate how 15 can be subtracted from 28 in the Characteristic method.

Solution :

	Decimal form	Binary form	Characteristic form
Minuend	28	011100	111100
Subtrahend	−15	101111	010001
Adding the characteristic forms, the result is			001101
Since the Sign Bit is not present in the result, the result itself gives the difference			001101 = 13

EXAMPLE 4.13 Show how 27 can be subtracted from 17 by the Characteristic method.

Solution :

	Decimal Form	Binary Form	Characteristic Form
Minuend	17	010001	110001
Subtrahend	–27	111011	000101
Adding the Characteristic form the result is			110110
Since the Sign bit is present in the result, 2's complement it to get the difference			101010 = –10

4.4 MULTIPLICATION BY THE ALU

The ALU uses 2 adjacent General Purpose Registers for performing Multiplication, since the result or the PRODUCT will be about double the size of either the MULTIPLICAND or the MULTIPLIER. Both the Multiplicand and the Multiplier will be in the Binary form. The Multiplication is performed by successive Additions and shifting Right the data in the Registers by 1 bit. Besides the 2 adjacent Registers, another Register is to be used for storing the Multiplicand. Also a Counter has to be used to keep the number of Right-Shifts.

The detailed procedure will be as follows—

(i) Initialise a Counter to keep track of the number of Right-Shifts.

(ii) Load the Multiplicand in a General Purpose Register.

(iii) Choose 2 other Registers which are adjacent to each other.

(iv) Initialise the Higher Order Register (HOR).

(v) Load the Multiplier in the Lower Order Register (LOR).

(vi) Check the Least Significant Bit (LSB) of the LOR.

If it is one, add the Multiplicand to the contents of the HOR. Increment the Counter by 1. Go to step (viii).

(vii) If the LSB of LOR is zero, increment the Counter by 1.

(viii) Check the Counter. If the number in it is less than the number of bits in any one Register, go to step (vi). Else, stop.

The Product will be available in the 2 adjacent Registers.

Example 4.14 will illustrate this.

EXAMPLE 4.14 Assuming a Computer with Registers of 4 bits only, show how the multiplication of 13 and 9 will be executed.

Solution :

Multiplicand = 13 = 1101
Multiplier = 9 = 1001

(The procedure is illustrated in Table 4.5)

TABLE 4.5

Product = $(01110101)_2 = (117)_{10}$

HOR	LOR	Counter	Operation
——	——	0	Initialise the Counter.
——	——		Choose 2 adjacent registers.
0000	——		Initialise HOR.
0000	1001		Load Multiplier in LOR.
			Check the LSB of the LOR.
1101			Since it is 1, add the Multiplicand to HOR.
1101	1001		Shift right the data in both the registers by 1 bit.
−110	1100		
		1	Add 1 to the counter.
			Since number in the counter is less than 4, Check the LSB of LOR.
			Since it is 0, shift right.
− − 11	0110		
		2	Add 1 to the counter.
			Since the number in the counter is less than 4, check the LSB of LOR.
			Since it is 0, shift right.
− − −1	1011		
		3	Add 1 to the counter.
			Since the number in the counter is less than 4, check the LSB of LOR.
1101	1111		Since it is 1, add the multiplicand to the HOR.
1110	1011		
−111	0101		Shift right.
		4	Add 1 to the counter. Since it is now equal to 4, stop.
0111	0101		The product is the data in both the registers.
PRODUCT			

4.4.1 Multiplication Circuit

In Multiplication circuit also, a Serial Adder is used. (Refer Fig. 4.11 in the next page) R_0 and R_1 are the adjacent registers used. Out of these, R_0 is the HOR and R_1 is the LOR. The Multiplicand is stored in the register R_2. The Multiplier is stored in R_1. A RIGHT-SHIFT signal is used to shift the bits of R_0 and R_1 by a single bit each time to the right.

If the rightshifted bit from R_1 is 1 the Multiplicand and the contents in R_0 are added by the Full-Adder and the sum is each time sent to R_0. If it is 0 the addition is not performed. This is followed by the Right-shift and so on. A counter is used to keep track of the Right-Shifts and when the count in it reaches the value equal to the bits in the register, the operation is terminated and the Product is available in registers R_0 and R_1.

4.5 DIVISION IN COMPUTERS

Division in computers is performed by successive Shift-Left and subtract operations. The Subtraction may be of 2's complementing method. For performing division also, 2 adjacent registers are to be used. The DIVIDEND is loaded in these 2 registers. The Divisor is stored in another register. The steps by which division is executed are given.

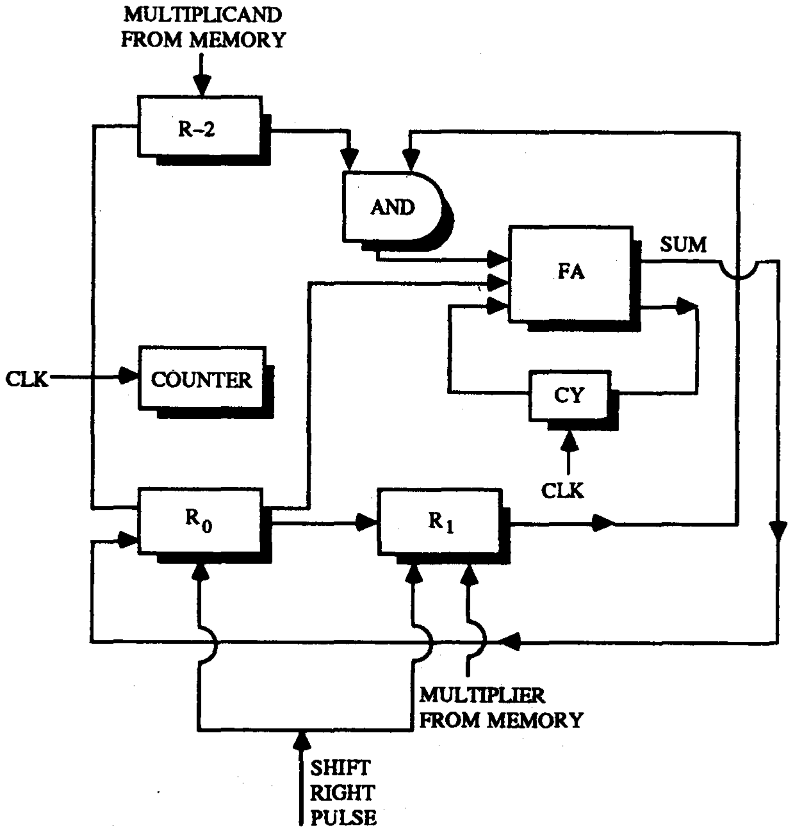

MULTIPLICAND FROM MEMORY
R-2
AND
FA
SUM
CLK
COUNTER
CY
CLK
R_0
R_1
MULTIPLIER FROM MEMORY
SHIFT RIGHT PULSE

Fig 4.11 Multiplication Circuit

(i) Initialise a counter to keep track of the Left-Shifts.

(ii) Load the Divisor in a register.

(iii) Choose 2 adjacent registers and load the Dividend in them.

(iv) Shift left the contents of both the registers by 1 bit to the left.

(v) Increment the counter by 1.

(vi) Compare the contents of the HOR with the Divisor.

(vii) If the Divisor is less than or equal to the other, Subtract it from the data in the HOR. Post 1 in the LSB of the LOR. Go to step (ix).

(viii) Else, post 0 to the LSB of the LOR.

(ix) Check whether the count in the Counter is equal to the number of bits in any one register. If not, go to step (iv).

(x) If equal, stop the operation. The REMAINDER will be available in the HOR, and the QUOTIENT in the LOR.

Example 4.15 will illustrate this:

EXAMPLE 4.15 Show, how in a Digital Computer with 4 – bit registers, the division of 61 by 9 wil be done.

Solution : The steps in the Division are given in the Table 4.6.

Dividend: 61 00111101
Divisor: 9 1001

HOR	LOR	Counter	Operation
——	——	0	Initialise the counter
0011	1101	0	Load dividend in 2 adjacent registers
0111	101–	0	Shift left the contents of both registers
0111	101–	1	Increment the counter by 1
			Compare divisor with data in HOR.
0111	1010		Since divisor is greater, post 0 to LSB of LOR
1111	010–		Shift left data in 2 registers
1111	010–	2	Increment counter by 1.
			Compare divisor with data in HOR.
1001			Since divisor is less, subtract it from HOR.
0110	010–	2	
0110	0101	2	Post 1 to the LSB of LOR
1100	101–	2	Shift left
1100	101–	3	Increment the counter by 1. Compare the divisor with data in HOR
1001			Since the divisor is less, subtract it from HOR
0011	101–	3	
0011	1011	3	Post 1 to LSB of LOR
0111	011–		Shift left
0111	011–	4	Increment counter by 1
			Compare divisor with HOR
0111	0110	4	Since the divisor is less, post 0 to LSB of LOR
REMAINDER	QUOTIENT		Since the counter now shows 4, which is the size of any register, stop the operation. The content of the LOR viz. 0110 will be the QUOTIENT and the contents of the HOR viz. 0111 the REMAINDER.

Dividend: 00111101 61 }
Divisor: 1001 9 }

Quotient: 0110 6 }
Remainder: 0111 7 }

4.5.1 Divider Unit

Fig. 4.12 gives the Block Diagram of a Divider Unit. The Registers R_0 and R_1 are 2 adjacent registers between which the left-shift is possible. The Dividend is moved into them from the memory. The Divisor is moved from the memory and stored in the register X. The data in the registers R_0 and R_1 are shifted left each by one bit and the contents of R_0 are compared with the Divisor in the register X. A device called as COMPARATOR is used for this comparison (refer Fig. 4.12). If the Divisor is less than or equal to the other, it is subtracted from R_0 at R_0 and a 1 is posted at the LSB of R_1. Otherwise a 0 is posted. The above procedure of Left-shift etc is continued until the counter which keeps track of the left-shifts equals the number of bits in any one register. Then the operation is stopped. The Remainder will now be available in R_0 and the Quotient in R_1.

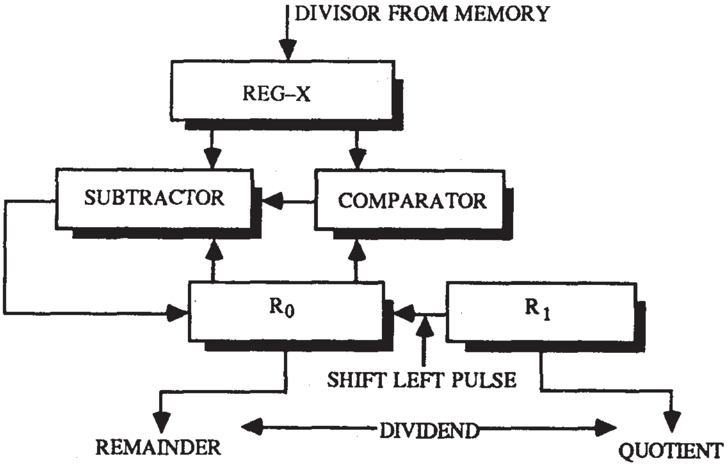


Fig 4.12

4.6 LOGICAL OPERATIONS

In the modern digital computers, the logical operations are performed with the help of the FLAGS in the Status Register which is also called Conditional Code Register or Flag Register. These flags are Binary Flags; that is each of them can store either 1 or 0. If a flag carries 1, it is said to be SET. If it carries 0, it is said to be CLEAR.

The important Flags in any Status Register are (i) Sign Flag – S, (ii) Overflow/Underflow Flag – V, (iii) Zero Flag – Z, and (iv) Carry Flag – C (refer Fig. 4.1). The flag S will be set if the data handled by the ALU is negative. The flag V will be set if there is an overflow or underflow condition. If the numeric data is too large to be handled by the processor, it is called Overflow Condition. If it is too small, it is called Underflow Condition.

The flag Z will be set if the data handled by the ALU or the Result is Zero. The flag C will be set if there is a resulting CARRY after an Addition or if there is a resulting BORROW after a Subtraction.

Logical operation involves Comparison of 2 data elements, say, A and B to find whether A is greater than or less than or equal to B. This is done by subtracting B from A and checking the flags Z and C. If the Z flag is set, it means that A equals B. If the flag C is set, it means that A is less than B. Otherwise, A is taken to be greater than B. Normally, a compare/logical operation will be followed by a Branch/Jump instruction.

Old type computers used a device known as COMPARATOR for comparing two variables or a variable with a constant.

Summary

The ALU in the digital computer contains circuits to perform Arithmetic and logical operations. The Binary Flags in the Status Register are useful in performing logical operations. All Arithmetic operations are performed with the numeric data in the Binary form. Subtraction is done mostly by the 2's complementing method. Multiplication is done by a process of Addition and shifting right. Division is done by a process of shifting left and subtraction.

Exercise/Questions

4.1 What are the functions of (i) ALU, (ii) The Status Register, (iii) The General Purpose Registers, and (iv) the Clock in a digital computer?

4.2 Explain why the binary form is preferred for performing Arithmetic operations on numbers, in a digital computer?

4.3 Differentiate between a Half-Adder and a Full-Adder. Give their Truth Tables.

4.4 Illustrate with suitable examples—(i) Subtraction—Direct method, (ii) Subtraction by 1's complementing method, (iii) Subtraction by 2's complementing method, and (iv) Subtraction by the Characteristic Method.

4.5 Explain with Block Diagrams—(i) A Ripple Carry Adder, (ii) An Adder-cum-Subtractor, (iii) A Multiplier circuit, and (iv) A Divider circuit.

4.6 Assuming a Digital computer with 8 bit Registers, show how the multiplication of 148 by 126 will be performed.

4.7 Show how the division of 18945 by 198 will be performed in the above computer.

for NOTES

Chapter Five

The Control Unit

5.1 INTRODUCTION

The Control Unit is a part of the CPU. It is also called The Control and Timing Unit or The Control and Sequencing Unit. It regulates all the basic operations in the computer system. For this, it generates Synchronisation signals. It uses inputs from a Master-Clock to generate the above signals (Fig. 5.1).

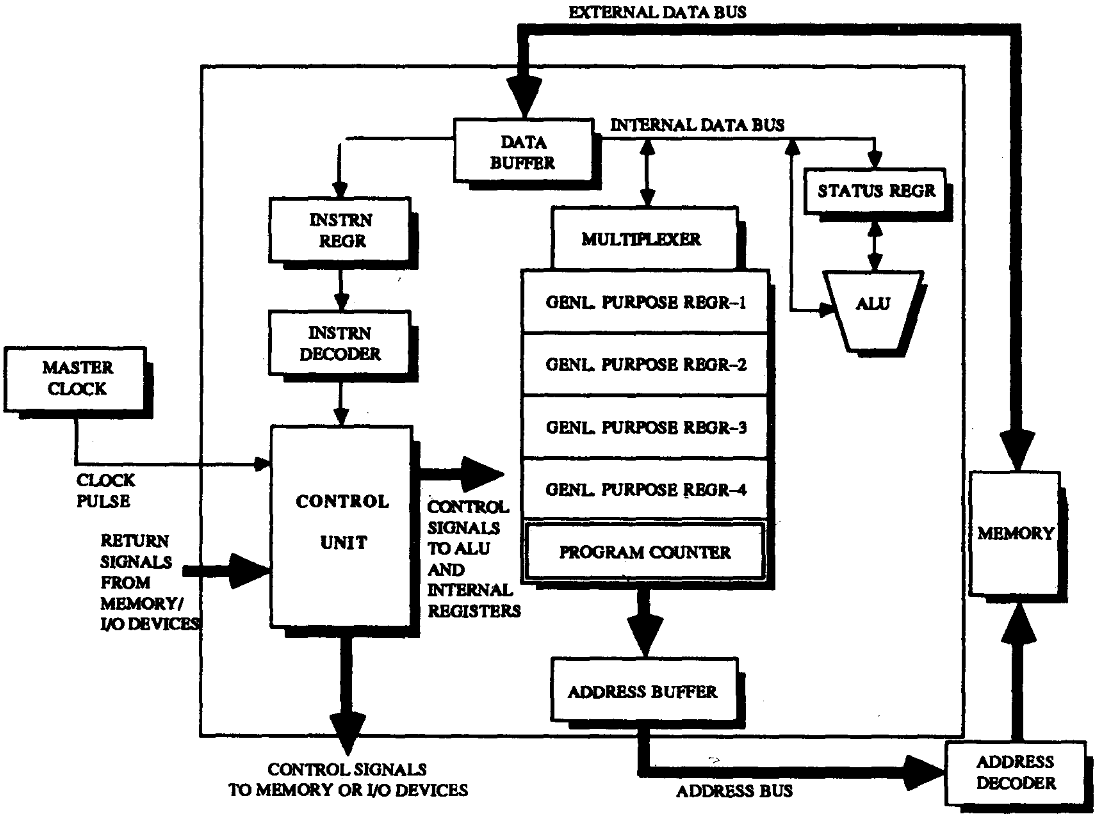

EXTERNAL DATA BUS
DATA BUFFER
INTERNAL DATA BUS
STATUS REGR
INSTRN REGR
MULTIPLEXER
ALU
GENL. PURPOSE REGR-1
GENL. PURPOSE REGR-2
GENL. PURPOSE REGR-3
GENL. PURPOSE REGR-4
INSTRN DECODER
MASTER CLOCK
CLOCK PULSE
CONTROL UNIT
CONTROL SIGNALS TO ALU AND INTERNAL REGISTERS
RETURN SIGNALS FROM MEMORY/ I/O DEVICES
PROGRAM COUNTER
MEMORY
ADDRESS BUFFER
ADDRESS DECODER
CONTROL SIGNALS TO MEMORY OR I/O DEVICES
ADDRESS BUS

Fig 5.1

5.2 PROCESSING OPERATION

Data processing in a digital computer is of 2 major operations. They are called—(i) Opcode Fetch, and (ii) Execute. Each of them is also called a Machine-Cycle.

The user's program and the data to be processed by it are stored in the memory unit. Taking the program instruction from the memory and transferring to the Instruction Decoder in the CPU is called the Opcode-Fetch operation.

The Execute cycle involves—(a) decoding the instruction, (b) bringing the data either from the memory or from the Input Device, and (c) executing the decoded instruction on that data.

5.2.1 Opcode-Fetch Cycle

There is a special Register in the CPU, known as the Program Counter (refer Fig. 5.1). This is designed to give the address of the next instruction to be fetched from the memory. This address is taken by the Address buffer. Then the address in the Program Counter gets increment automatically to indicate the address of the next instruction.

A set of lines called the ADDRESS BUS links the Address Buffer with the Address Decoder in the memory unit. Each line is called an Address Bit. The number of the Address Bits depends upon the size of the Memory. If the memory has L locations, $L = 2^n$, where n is the number of bits in the Address Bus. The Address Decoder decodes the address carried by the Address Bits and selects the location in the memory.

Another set of lines is used to connect the memory unit and the Data Buffer in the CPU. This is called the Data Bus. This is used to transfer the data between the CPU and the memory as well as between the CPU and the I/O devices. In this case, it transfers the instruction in the Memory location selected to the Data Buffer in the CPU. The Instruction Register takes the instruction from the Data Buffer and passes it to the Instruction Decoder.

5.2.2 Executive Cycle

The Instruction Decoder decodes the Instruction, (ie,) it divides each instruction into a number of smaller instructions which are called Micro-instructions. For example, if the decoded operation is Subtraction, it will split it into the following micro-instructions:

(i) Bring the Minuend from the memory and store it in a general purpose register.

(ii) Bring the Subtrahend from the memory and store it in another general purpose register.

(iii) 2's complement the subtrahend.

(iv) Add the above with the minuend.

(v) Store the result in either of the above 2 registers.

(vi) Check whether the sign bit is 1.

(vii) If so, 2's complement the result.

5.3 TYPES OF CONTROL

There are 2 methods by which the control unit can generate the control signals. They are—

(i) Hard-wired control, and

(ii) Micro-programmed control.

5.3.1 Hardwired Control

In this method of control, the micro-instructions are generated in the form of signals in proper sequence. These signals are synchronised with the Timing Signals from the Master-Clock using the devices known as AND-GATES (refer Fig. 5.2). The output signals from the AND-GATES are called the control signals and are carried to the respective devices through the control lines which form the CONTROL-BUS. On receipt of the control signal, the memory or the ALU or the General Purpose Register etc executes the relevant Micro-operation.

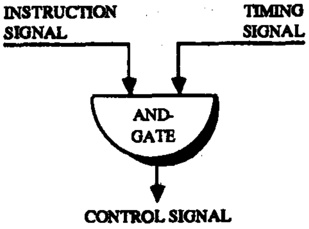


Fig 5.2

5.3.2 Micro-programmed Control

In this method, all the instructions that can be executed by the computer, such as ADD, SUBTRACT etc are listed. The operation to be executed by each such instruction is divided into a number of micro-operations. Programs are prepared with instructions to execute these micro-operations. These instructions are called MICRO-INSTRUCTIONS and the programs containing them are called MICRO-PROGRAMS. There will be a separate Micro-Program for each instruction like ADD, SUBTRACT etc. These Micro-Programs are stored in a specific area in the memory which is called the MICRO-PROGRAM MEMORY. The Addresses of the memory locations in which these micro-programs are stored are held in the Control Unit (refer Fig. 5.3).

When the Instruction Decoder decodes the type of the instruction and passes the information to the control unit, it transfers the memory address of the related micro-program to a device called the MICRO-PROGRAM-COUNTER. The Address Bus takes this address and accesses that location. The instructions in that micro-program are taken to the processor one by one, each in the form of a CONTROL-WORD and executed.

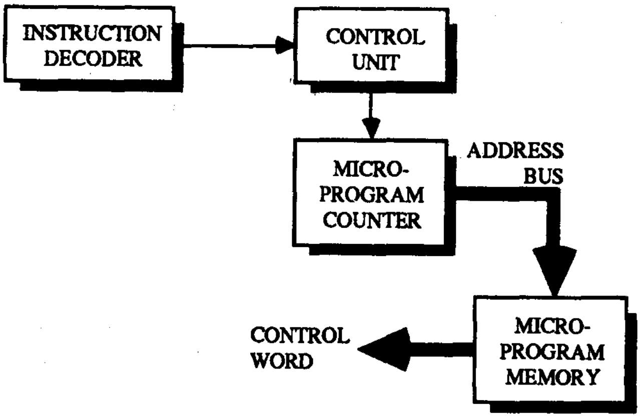


Fig 5.3

The Micro-programmed control is preferred to the Hardwired control, since any new instruction can be added to the system by preparing a micro-program for it and storing it in the micro-program memory. This would involve no addition or no alteration to the existing hardware. Also, any existing instruction can be modified or deleted without affecting the hardware. The modern computers, therefore, adopt the micro-programmed method of control only.

Summary

The control unit controls the input, processing and output operations in the computer system. These operations are executed by the programmed instructions stored in the memory of the computer. Their execution involves 2 major operations, namely, opcode fetching and, decoding the instruction fetched from the memory and executing it. The decoded instruction will be in the form of a group of micro-instructions which are executed by control signals to various devices. The control signals can be generated by hardware (hardwired control) or by software (micro-programmed control). Modern computers adopt the micro-programmed method of control.

Exercise/Questions

5.1 Describe the function of the control unit in a digital computer.

5.2 What are the two major-operations under processing?

5.3 Give the micro-operations under the Opcode-Fetch cycle.

5.4 Describe the Hardwired control and the Micro-programmed control. Which of these is preferable, and why?

for NOTES

Chapter Six

Input and Output Devices

6.1 INTRODUCTION

The purpose of the Input Device is to feed the Data to be processed and the Program to the computer system. The purpose of the Output Device is to display the results of processing. The input and the output devices are together known as I/O Devices.

6.2 INPUT DEVICES

An input device is to transfer data or instruction recorded on an input medium like Card, Magnetic Tape etc, to the memory of the computer.

6.2.1 Characteristics of an Input Medium

An input medium like the Card or Floppy Disc etc. should have the following characteristics—

(i) We should be able to record alphabets, digits and other characters such as $+, -, \times$ etc. in a coded form.

(ii) It should be easy to transfer the data recorded in the input medium to the memory of the computer.

(iii) The data transfer should be as fast as possible.

6.2.2 Different types of I/O Media and Devices

The different types of input media that are used with a computer are—(i) Key-Board, (ii) Punched Card, (iii) Magnetic Tape, (iv) Cartridge Tape, (v) Magnetic Disk units, (vi) Floppy Disk, and (vii) other specialised forms of input devices. A device is associated with each such medium to transfer the data recorded on it to the computer memory. For example, a CARD-READER is required for 'reading' the Card, that is to transfer the data recorded on Card to the memory of the computer.

6.2.2.1 Key-Board

In case of small computers called Personal Computers (P.C.'s) and in case of mini-computers operated from a number of terminals, the Key-Board is used as the input device. Such computers contain 2 sections of memory called (a) Main Memory, and (b) Secondary Memory, respectively. The data entered through the key-board is stored in the secondary memory.

Most of the keys in the key-board are the same as in a standard typewriter. In addition, there are certain keys called the FUNCTION KEYS. These are useful in giving instructions such as 'Delete', 'Back-space' etc. to the computer directly. There are, in all, about 64 keys in a standard key-board (refer Fig. 6.1).

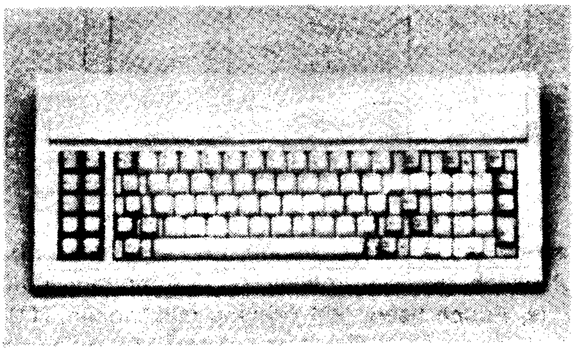


Fig 6.1 Key-Board

6.2.2.2 Punched Card

A few decades back, Punched Card was extensively used for recording program and data. A Card is of length $7\frac{3}{8}$ inches, width $3\frac{1}{4}$ inches and thickness 0.007 inch. It has 80 vertical columns that are numbered 1, 2, 3, etc, 80 from left to right. There are 12 horizontal rows that are numbered 12, 11, 0, 1, 2, 3, etc. 9. Though there are 12 columns, only one line of data can be recorded in a Card. So a Card can contain the details of one person or one material or one set of readings only. So it is called a UNIT RECORD. In one column of a Card only one character can be recorded. It will be in the form of rectangular punches in one or more rows (refer Table 6.1). As such, a Card can contain a record of maximum 80 characters only. The device used for punching data on a Card is called a CARD-PUNCH (refer Fig. 6.3). It has a key-board using which data can be entered on the Card. Since the data is entered without the use of the computer, it is called OFF-Line data entry

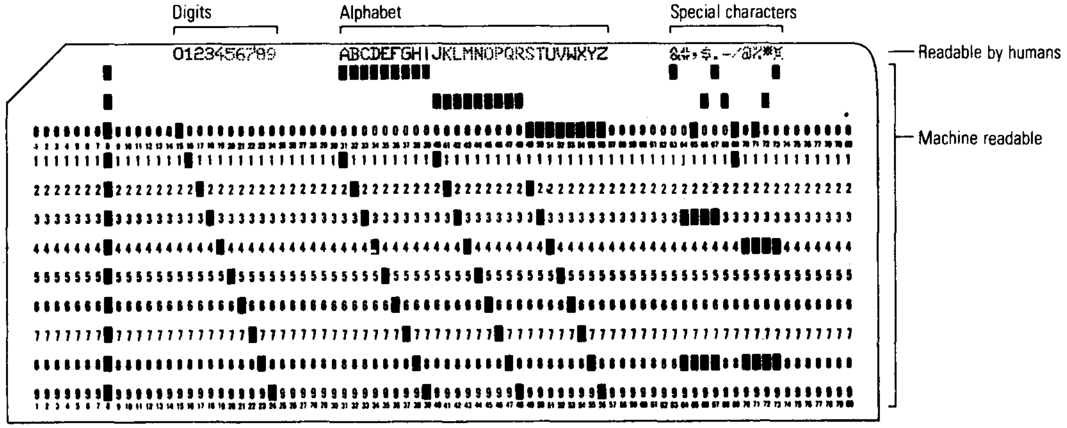


Fig 6.2 Punched Card

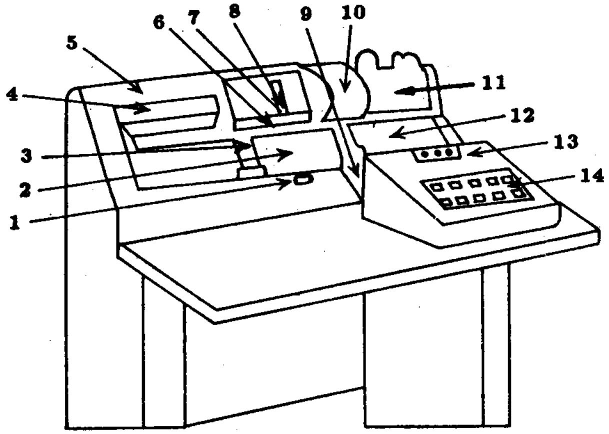


Fig 6.3 An Off-Line Card-Punch
(1) Back space key, (2) Read-station, (3) Read head, (4) Stacker, (5) Control switch, (6) Control lever, (7) Column indicator, (8) Program unit, (9) Punch head, (10) Printing mechanism, (11) Hopper, (12) Punching station, (13) Function switch, (14) Key-guard.

Table 6.1 gives the list of characters and their punching configurations on the Card. They are called Hollerith's Code.

TABLE 6.1 HOLLERITH CODE

Character	Punch at Rows	Character	Punch at Rows
1	1	O	11,6
2	2	P	11,7
3	3	Q	11,8
4	4	R	11,9
5	5	S	0,2
6	6	T	0,3
7	7	U	0,4
8	8	V	0,5
9	9	W	0,6
A	12,1	X	0,7
B	12,2	Y	0,8
C	12,3	Z	0,9
D	12,4	+	12
E	12,5	−	11
F	12,6	*	11,4,8
G	12,7	/	0,1
H	12,8	=	3,8
I	12,9	(	0,4,8
J	11,1	)	12,4,8
K	11,2		12,3,8
L	11,3	, comma	0,3,8
M	11,4	' quote	4,8
N	11,5	$	11,3,8

A Card-Reader is to be used for the transfer of data from the Card to the Memory of the Computer (refer Fig. 6.4). It has a 'Reading-Station' which has a circuit of 960 pairs of brushes corresponding to the 960 possible punches in the Card. When a Card passes through a reading station, in positions where there is a punch, contact is established between the corresponding pair of brushes and so a current will flow in the circuit. This will energise the relevant bits in the corresponding location in the Memory. About 1,000 Cards will thus be read by the Card-reader in a minute. This rate of data transfer is very slow compared to the other I/O devices. Also, punched Cards are not reusable. They require large storage space. As such, they are not much in use nowadays.

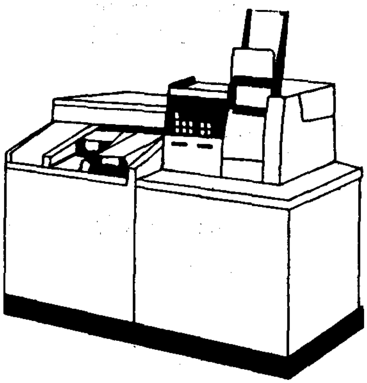


Fig 6.4 Card Reader

6.2.2.3 Magnetic Tape

The magnetic tape is colloquially referred MAG-TAPE. It is a plastic tape of length 2,400 feet, 0.5 inch width and 0.0015″ thickness. One side of it is coated with 9 tracks of Ferric oxide (Magnetic Material) all along its length. The tape is mounted on two spools each of 10.5″ diameter. It is run at a speed of 100 IPS (inches per second). The Tape Unit contains a Write-Head to record data on the tape. A Key-Board is used to enter data on the tape. The data entry is to be made with the computer 'on'. Such entry is called ON-LINE data entry.

Depression of the key on the key-board produces combination of voltages. The Write-Head converts these into Magnetic fields that are recorded on the top eight tracks of the tape. The recording will be either in the ASCII code or in the EBCDIC code. The 9th track functions as the PARITY TRACK. It gets automatically energised (magnetised), if otherwise the parity is even. The recording density on tape is 1,600 characters per inch. A data record, which can be recorded in an 80 column Card of $7\frac{3}{8}$ inch length, can be recorded within a length of 0.05″ on a Mag-Tape. A Read-Head is provided to transfer data on tape to the computer memory.

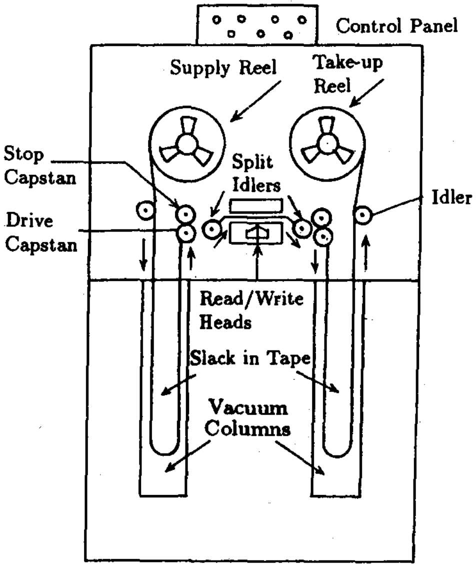


Fig 6.5 Mag-Tape Unit

The disadvantage in the mag-tape is, while recording data, 0.375″ of tape is skipped before writing a record and another 0.375″ after writing a record. As such, there will be a gap of 0.75″ in between adjacent records. This gap is called IRG (Inter Record Gap). To minimise the wastage of tape area in the form of IRG's, the records are written in blocks of 10 or 15 or 20 etc... 50. This is referred to 'Blocking of Tape Records'. The number of records in each block should be the same. The number of records in a block is called the BLOCKING FACTOR. The Block should not contain too many records since the computer has to read a block of records at a time and if the block-size is too large a large area in the memory has to be used for 'Reading'.

Even after allowing for the IRG's between blocks, the Mag-tape can store. A large number (about 2 million) of records. Thus, it offers a compact storage of records. The data transfer rate between the tape and the memory is as high as 160 Kilo bytes per second. So the Mag-tape is used both as input and output media. It can also be used as a Secondary Storage Device; but, since only serial access is possible in it, the Magnetic Disk which has the direct access facility is preferred to the tape as a secondary memory.

6.2.2.4 Cassette Tapes

Special digital Cassette Tapes are now available for recording and storing input data to the computers. Compared to the Mag-tape, the cassette tape has a low recording density. It is of length 300 feet only. But, since it does not have the IRG, it can store 700 thousand characters (storage capacity 700 Kilo Bytes). The cassette tape has a data transfer rate of 700 characters per second.

Just as Mag-tape, the Cassette tape permits serial access only; that is, records can be accessed one by one only. So it is used as an Input/Output medium only. But both magtape and the cassette tape are often used as BACK-UP storage.

6.2.2.5 Magnetic Disk

The Magnetic Disk is also known as Winchester Disk or Hard-Core Disk. A Disk Unit contains a number of circular plastic disks of diameter 8″. The number of disks depends on the storage capacity of the disk unit. For example, an 80 Megabyte disk unit will contain 6 disks. Normally, data can be recorded on both sides of the disk. But the upper surface of the topmost disk and the lower surface of the bottom-most disk are covered with End-Plates: So, in a disk unit of 6 disks, only 10 surfaces are available for recording data (refer Fig. 6.9).

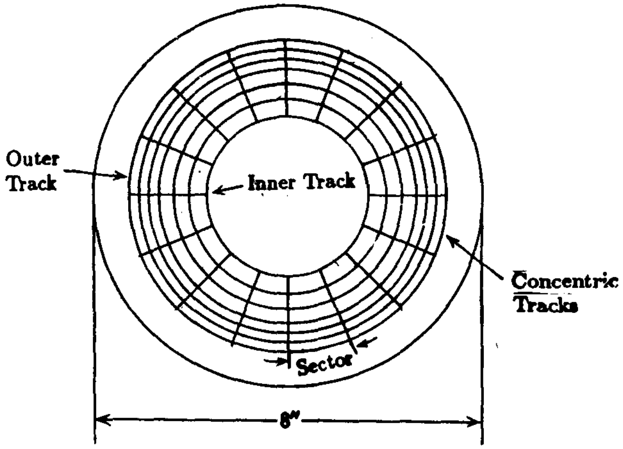


Fig 6.6 Disk Surface

The disks are mounted on a spindle and are placed within an evacuated plastic container. Such a container is called a Disk-Pack. (Fig. 6.7). The spindle is driven by a Disk-Drive at 3,600 R.P.M. Each disk-surface contains about 512 concentric tracks of magnetic material. Each track is divided into about 32 sectors (Fig. 6.6). Data is recorded with one record per sector. Each sector can hold maximum 512 characters (bytes).

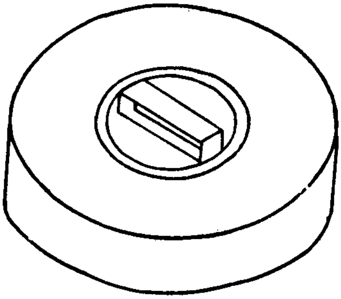
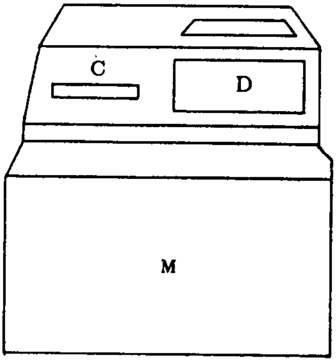


Fig 6.7 Disk Pack



(C) Control switches
(D) Disk mount
(M) Driving motor cabinet

Fig 6.8 Disk Drive

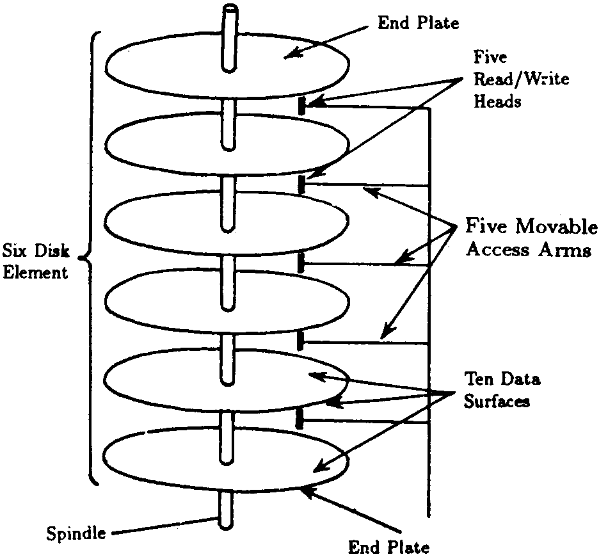

End Plate
Five Read/Write Heads
Five Movable Access Arms
Six Disk Element
Ten Data Surfaces
Spindle
End Plate

Fig 6.9 Disk Unit

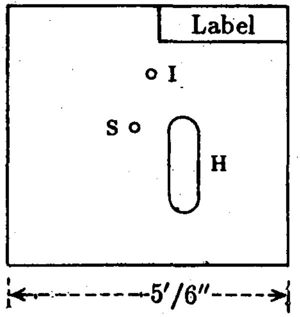
A Radial arm is provided in between adjacent disks. It has 2 sets of Read and Write heads, one set for the disk surface above and the other for the disk surface below. Each of the 163,840 sectors in the disk unit is assigned a distinct address from 000000, 000001, 000002, etc... 163839. The disk unit is so designed that any sector in it can be accessed at random and data can be written in or read from it. This is called RANDOM ACCESS or DIRECT ACCESS. In view of the direct accessing facility, the disk is used to store files whose records are to be processed at random. Also, it is used as the Secondary Memory (refer Chapter 7). The data transfer rate from the disk is 5 mega BITS per second.

6.2.2.6 Floppy Disk

The floppy disk is a single plastic disk of diameter 8″. It has its surfaces coated with 77 concentric tracks of magnetic material called 'Micro-soft Magnetic material'. So it is also referred as Micro-soft (M.S.) disk. Each track in the M.S. Disk is divided into 32 sectors and in each sector maximum 128 bytes of data can be recorded. A disk can store 720 kilobytes of data. The floppy Disk is driven by a drive at a speed of 360 R.P.M. The data transfer rate is about 250 Kilo bytes per second. (refer Fig. 6.11).

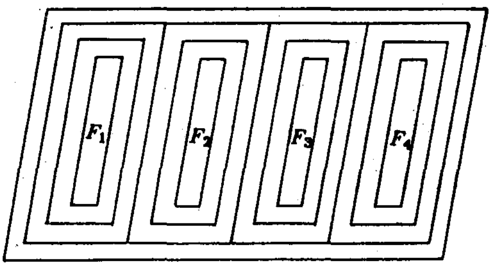
Nowadays, instead of the 8″ floppy disks, smaller size disks known as DISKETTES, which are of $5\frac{1}{4}''$ diameter, are in use.

Since the tracks in the Floppy Disk or the Diskette are of microsoft material, it is placed in an cardboard envelope to prevent wear and tear (refer Fig. 6.10). A slot known as Head-Slot is provided for the Read/Write head to access the tracks. Also, both in the Winchester disk and in the floppies, the tracks will be in a strip which is more or less at the middle of the disk surface (refer Fig. 6.6). The number of tracks per inch in that strip is called TRACK DENSITY. The normal value of the track-density is either 48 Tracks per inch (TPI) or 96 TPI. The floppy disks, and more especially the Diskettes, are widely used nowadays with the PC's. They are used (i) for entering and storing user's program and data, and (ii) as BOOT-FLOPPY, ie, for storing the Operating System and other Systems Programs. (The operating system stored in a Floopy is called as MSDOS, ie, Micro-Soft Disk Operating System.



I : Index Hole
S : Spindle Hole
H : Head Slot

Fig 6.10



F_1 F_2 F_3 F_4 : Floppy-Drive

Fig 6.11

6.2.2.7 Input Devices of Specialised Forms

These include—(i) OCR, (ii) OMR, (iii) Writing Pad Device, (iv) Voice Recognition System, (v) MICR, and (vi) Bar Codes.

6.2.2.7.1 OCR

OCR means OPTICAL CHARACTER RECOGNITION. For this, an OPTICAL READER is used. It recognises alphabets, digits and other characters typed in special sheets of size $8\frac{1}{2}$ inches by 11 inches and transfers them to the computer memory. The OCR unit consists of a Scanner, a Recognition Unit and a Decoder (refer Fig. 6.12).

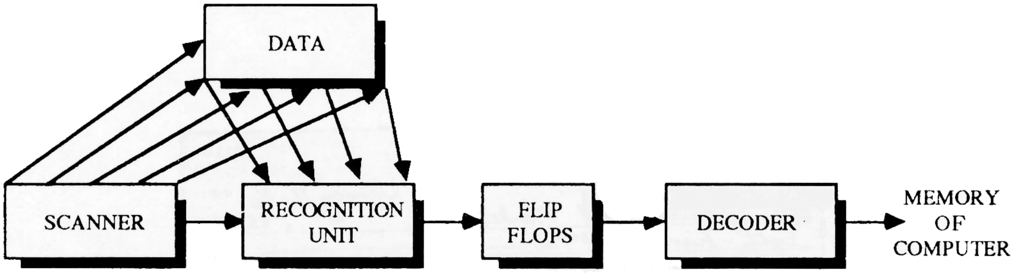


Fig 6.12 OCR

The scanner is a source of light. The light given out by it is beamed on the sheet, the data recorded on which is to be read. The light falling on the sheet is reflected on to the Recognition Unit. It consists of an array of photo-cells arranged in the form of a matrix. These cells are connected to a set of flip-flops. Whichever photo-cell receives light reflected from the sheet will send a charge to the flip-flop connected to it. The Decoder converts these charges into ASCII or EBCDIC code and transfers it to the Computer memory.

6.2.2.7.2 OMR

OMR means Optical Mark Reading and Recognition. This also contains an Optical Reading Device. This transcribes the mark made by a lead-pencil into a stream of electronic pulses which are converted into ASCII or EBCDIC code and transferred to the computer or its memory. This is made use of, for example, in evaluation of answers to multiple choice questions wherein answers are ticked in one of the rectangular boxes with a lead-pencil.

6.2.2.7.3 Writing-Pad Device

This device can read Alphameric data written by hand. But it can be used only with a special type computer known as RECOGNITION PROCESSOR. This processor builds up the formation of any hand-written character by use of x and y co-ordinates and stores them as such. This is useful in investigation of forgeries as also for routine signature verifications in the banks.

6.2.2.7.4 Voice Recognition System

This converts the data dictated through a microphone into pulses and transfers them to the computer system/memory which stores them. The data stored will represent the pattern of the speech. The disadvantage in this method is that only a limited terms or vocabulary can be stored.

6.2.2.7.5 MICR

MICR means Magnetic Ink Character Reader. It can read documents or records in which the words and figures are printed using a special Magnetic Ink which contains Iron Oxide. The MICR can transfer the characters into the computer memory. Since this guards against any unauthorised alteration in the entries, this is widely used in banks.

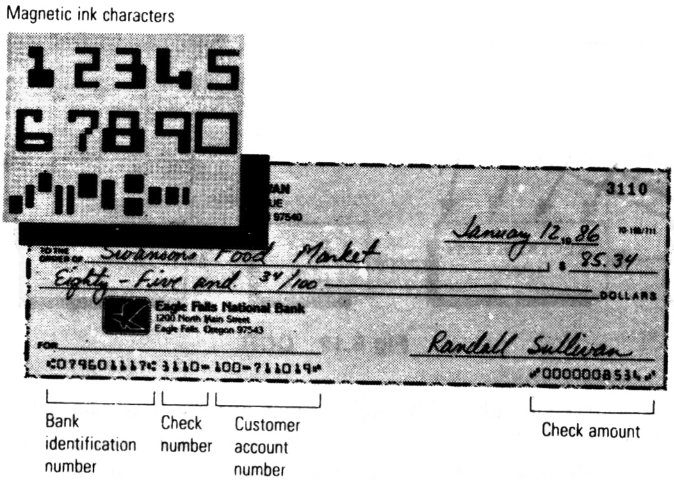

Magnetic ink characters
12345
67890
3110
January 12, 86
Swansons Food Market
85.34
Eighty-Five and 34/100
DOLLARS
Eagle Falls National Bank
1200 North Main Street
Eagle Falls, Oregon 97543
Randall Sullivan
Bank identification number
Check number
Customer account number
Check amount

Fig 6.13

6.2.2.7.6 Bar Codes

BAR CODES are used extensively to denote the type or category of an item and the rate or cost. About 10 or 11 bars of different thicknesses and varying space are used for this purpose. In case of books, these are printed in the back-cover. In case of other articles, they are printed in the packets containing them (refer Fig. 6.14).

31693

0 70999 00995

ISBN 0-345-31693-2

Fig 6.14 Bar Code

6.3 OUTPUT DEVICES

The main purpose of an Output Device is to display the result of the processing in a form that is easily readable to the user (programmer). The Card or tape or disk can be used as an output media. But the information stored in them cannot be easily reckoned. As such, the most used output devices are (i) the Screen, and (ii) the Printer.

6.3.1 The Screen

The Screen is also known as the VDU (Video Display Unit) or the MONITOR. It can display 24 lines of data at a time. Each line can contain a maximum of 80 characters inclusive of blanks, if any. The screen is the expanded part of a CRT (Cathode Ray Tube). It has its own memory of 2K (2 kilo bytes) to store the characters to be displayed. This memory is of core type. The information to be given as output is sent from the computer to this memory in the Binary form. A device known as CRT interface takes them and sends to a device known as DOT-MATRIX CHARACTER GENERATOR. This converts the output data into Characters which are formed by dots of 7 rows and 5 columns and transfers them in the form of analog signals to the CRT unit. The CRT unit contains the standard vertical and horizontal sweep deflection circuits. So, an electron beam sweeps across the screen several thousand times per second and thus the characters are displayed on the screen. A continuous display is achieved by refreshing it 60 times per second.

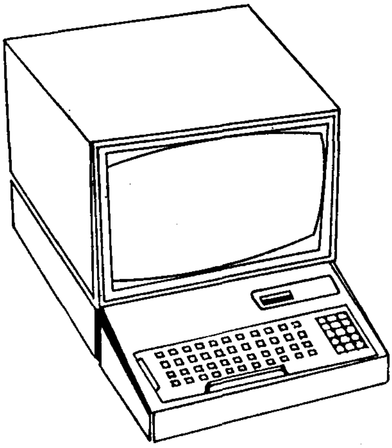


Fig 6.15 Screen with Key-board

In almost all cases, the Screen goes with a key-board (refer Fig. 6.15). This arrangement provides a sort of an interaction between the user and the computer. When a program or data is entered through the key-board the screen displays the entries. This facilitates the verification of the entries and correction of errors, if any. This is besides the displaying of the result of processing.

The VDU's are of 2 types, namely—(i) Monochrome monitor, and (ii) Colour monitor. The Monochrome monitor can display the output only in one colour, say, green or amber or violet. The colour monitor can give the display in 16 colours.

The monitor has the advantage that it can display Graphs and Bar Charts besides the alphameric data. But the defect is that it cannot give out a 'Hard-Copy' of the output like the Print-out.

6.3.2 Printers

The various categories of the printers are—(a) Impact Printers, (b) Laser Printers, and (c) Incremental Plotters.

6.3.2.1 Impact Printers

There are 3 types of Impact Printers, namely—(i) Dot-Matrix Printer, (ii) Line Printer, and (iii) Daisy-Wheel Printer.

6.3.2.1.1 Dot-Matrix Printer

The Dot-Matrix Printer uses a series of dots to form the characters. The Print-Head contains 9 × 7 array of pins. Depending upon the character to be printed, the pins arrange themselves and hit the inked-ribbon over the printing paper and it results in printing of character by character. The printing speed is 340 CPS (characters per second).

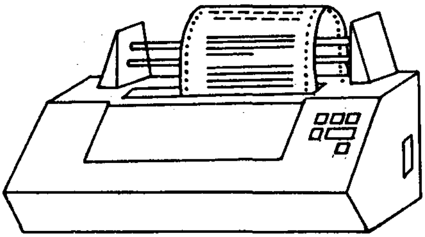


Fig 6.16 Dot-Matrix Printer

The Dot-Matrix Printer is not as fast as a Line Printer. Also, it cannot be worked for more than 2 hours at a time. But it is comparatively very cheap. Also, it can be made to print characters in any script.

6.3.2.1.2 Line Printer

A Line Printer is so called since it prints a line at a time. Line Printers that can print 80 or 120 or 132 characters per line are available. A Line Printer of 120 character capacity contains 120 revolving wheels mounted on a drum. Each wheel has all the alphabets, digits and other special characters embossed on its surface. An inked ribbon is placed between the wheels and the printing paper. The data for output from the computer is taken from the memory and assembled in the form of a line by means of revolving the wheels such that the corresponding characters are positioned along the striking line. A row of electrically operated hammers strike such that the impressions of the characters in the entire line is printed on the paper.

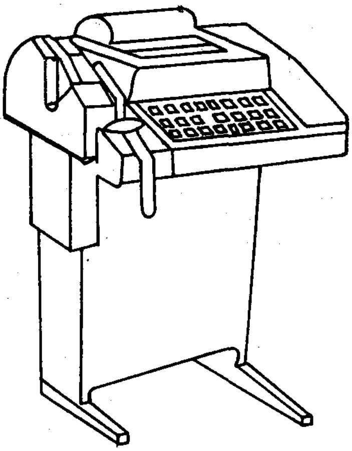


Fig 6.17 Line Printer

A Line Printer can print 600 lines per minute (LPM). It can be worked continuously for about 6 hours. So it is also called Heavy Duty Printer.

6.3.2.1.3 Daisy-Wheel Printer

A Daisy-Wheel Printer is also called 'Letter Quality Printer'. Its printing is just like that of an Electronic Type-writer. It contains a Daisy-Wheel. On its circumference all the alphabets, digits and other characters are embossed. These embossed characters are called FONTS. The Daisy-Wheel can rotate as well as move left to right and also from right to left. The wheel rotates such that the character to be printed is positioned on the striking line, and moves as such from left to right as it prints character by character just as a Dot-Matrix Printer. But it is comparatively slow, in as much as it can print only 40 characters per second (about 20 lines per minute).

6.3.2.2 Laser Printer

The Laser Printer is also known as a 'Non-Impact Printer' or 'Xerographic Printer'. It contains a photo conductive drum bearing all the characters. An electronically controlled Laser Beam traces out the character to be printed. Whichever character is selected by this beam, attracts an Ink Toner which transfers the image of that character on to a paper. This type of printer can print an entire page at a time. Its printing speed is about 10 pages per minute. But it is very expensive.

6.3.2.3 Plotter

Often the output information from a computer may have to be presented in the form of an ordinary or multi-dimensional graph or contour map. Such graphs or curves may be obtained on a screen. But this will not give the user a hard-copy of the curve. A device known as INCREMENTAL PLOTTER is used to give the hard-copy of the graphs and curves. It is a device with a large sheet of paper and a moving pen. The pen can be moved by a small increment of 0.005″ in any of the eight/four 45 degree directions. A Graphic Package program is to be provided to break the curve to be plotted into a series of such short line segments and the pen will describe these segments on the paper.

Summary

Any computer system should have one or more input device. The purpose of the input device is to transfer the user's program and related data from an input medium like a floppy disk to the computer memory. In the early days of computer evolution, Punched Cards were used for inputting data and program. Nowadays, Floppy diskettes are used along with the Personal Computers for the same purpose.

Big computers use Magnetic Tapes and Winchester Disks as input device.

The output device is to display the information or the result of processing. Screens are used with the PC's for this purpose. The screens can display Graphs also. But they cannot give a 'Hard-Copy' of the output; Printers are used to give the hard-copy. Dot-Matrix printers are used along with mini-computers and Line-Printers are used along with the Big Computers. Laser Printers have also now come into use. These printers are not capable of giving Graphical output. Incremental plotters are to be used for printing ordinary or multi-dimensioned graph.

In Business data processing, the outputs may be got in multiple media, say, in the form of a print-out plus a recording on tape and/or on disk which may be used later as input.

Exercise/Questions

6.1 What are the functions and the characteristics of—(a) the Input Devices, and (b) the Output Devices?

6.2 Briefly describe the following giving the salient features of each of them—

(i) Punched Card, (ii) Magnetic Tape, (iii) Floppy Diskette, (iv) Hard-Core Disk, (v) the Screen, (vi) the Dot-Matrix Printer, (vii) The Line Printer, (viii) the Laser Printer, and (ix) Daisy-Wheel Printer.

6.3 Write short notes on—(i) the Cassette Tape, (ii) OCR, (iii) OMR, (iv) Writing Pad, (v) Voice Recognition System, (vi) Bar Code, (vii) MICR, and (viii) Incremental Plotter.

for NOTES

Chapter Seven

Memory Units

7.1 INTRODUCTION

The Memory unit is an integral part of the Computer system. It is also called the STORAGE UNIT or STORE. As discussed earlier, its main function is to store the users' programs and related data.

7.2 MEMORY ADDRESS

The memory unit is made up of a number of cells. Each cell has a distinct label, which is called its ADDRESS. For example, a Memory of 2-K size will have 2048 (2 KILO) cells that will have addresses from 0 to 2047. The processor can place a data in any location of given address or take out data from it. Such a location is called a MEMORY LOCATION.

7.3 BIT, BYTE, WORD

The memory of a computer is designed to store data in the Binary form. That is, each character will be represented by a group of 1's and 0's. The element that can represent either a 1 or a 0 is called a BIT which is the abbreviation for 'Binary Digit'. There are 2 methods of representing any character in the Binary form, namely, ASCII and EBCDIC forms or codes. Both use an eight Bit code to represent any character. A combination of eight bits is called a BYTE. Each Byte can store a single character. A word is made up of 1 or 2 or 4 or 8 bytes. A WORD is the number of BITS or BYTES of data that can be transferred in a computer system by a single operation. Micro-Processor based computers may have a word-size of 1 Byte. The small computer PDP-11 had a word-size of 2 Bytes. The IBM 370 has a word size of 4 BYTES and supercomputers a word-size of 8 BYTES, that is, 64 Bits. The number of Bits in the Data Bus will be equal to the word-size of the computer

7.4 TYPES OF MEMORY

The present-day computers are provided with 2 memory devices, namely (i) the Main Memory, and (ii) the Secondary Memory (Fig. 7.1).

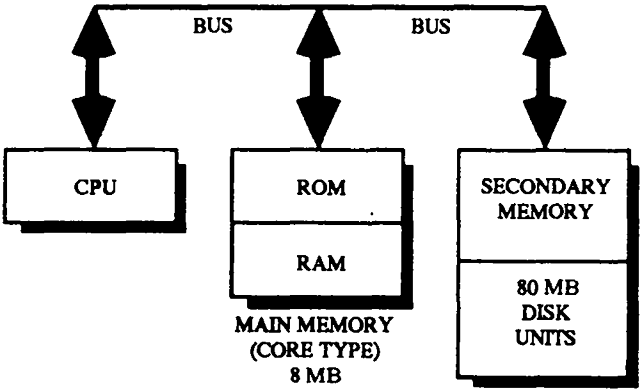


Fig 7.1

7.4.1 The Main Memory

The Main Memory is also known as 'Principal Memory' or 'Main Store' or 'Immediate Store' or 'Internal Memory'. It may be in 2 sections called (i) RAM, and (ii) ROM, respectively.

7.4.1.1 RAM

RAM means Random Access Memory. It is also called EQUAL TIME STORE. Any location can be accessed directly. The selection of a memory location is called Accessing. This is done by the ADDRESS BUS. The Address Bus is made up of a number of lines that are called Address Bits. The number of bits in the Address Bus depends on the size of the Memory. If there are *n* bits in the Address Bus, the memory will have 2^n addressable locations in all. Depending upon the configuration of the Address Bits, one of the memory-locations is selected/accessed. The purpose of the RAM is to store the users program and the related data. The RAM may be of Magnetic core type or MOS type.

7.4.1.1.1 Magnetic Core Storage

The core type Memory adopts FERRITE RING to represent a Bit. A Ferrite Ring is as small as a dot. Three wires pass through each ring (refer Fig. 7.2). They are called, respectively, X-wire, Y-wire and SENSE wire. A magnetised Ring (core) represents 1; else it represents a 0. To store 1 in any one of the rings, current, pulses of equal intensity, referred as half currents, are passed through the X and Y wires passing through that ring. The sense wire is used to 'Read' the data stored in that ring. If the ring through which it is threaded is magnetised, a current will be induced in it and this small current will be transferred to the Data Bus as binary 1. If the ring is in demagnetised state, no current will be induced in the sense wire and hence no current sent to the Data Bus which will be taken as binary 0.

The disadvantage of the Core type Memory is that it is Volatile, that is, its contents are erased when electricity is switched off.

7.4.1.2 MOS RAM

The RAM can be of Electronic Cores called MOS (Metal Oxide Semiconductors). Such a core is etched on to a Silicon Chip. The Binary 1 is stored as an Electronic charge in it. Absence of such a charge will indicate the storage of a 0. Memory using such memory-cells is called STATIC MEMORY.

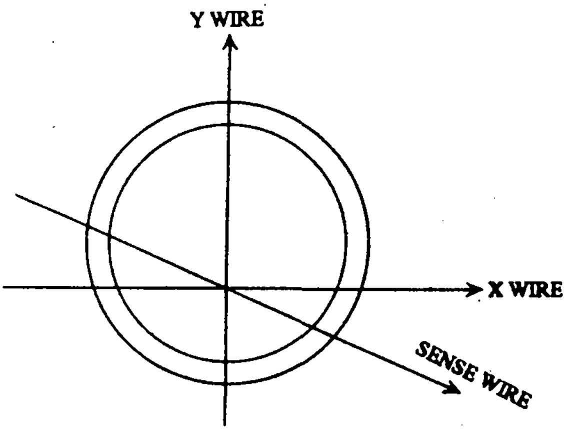


Fig 7.2 Core Ring

The advantages of the Static Memory over the Core type Memory are—(i) It is nonvolatile; that is, data stored in the memory will remain intact even if the power is switched off. (ii) The rate of Data Transfer from Static RAM is much faster than from the Core type Memory.

7.4.2 ROM

ROM means 'Read Only Memory'. It is so called since the user can make use of the systems Programs stored in it; but he or she cannot access it to store his or her program or data in it. This is said, 'Read Access is permitted in ROM, but not Write Access'. The memory cells in ROM are also of MOS.

ROM is used to store the (i) Operating System (O.S.), and (ii) Library Functions. These are prepared by the System Programmers of the manufacturers and stored more or less permanently.

The O.S. is a Supervisory Program that controls all the activities of the computer system. In the case of the small computers it is called CONTROL PROGRAM. In case of Micro-processors it is called a MONITOR PROGRAM.

The digital computer is essentially an ADDER. Subtraction is performed by complementing and adding. Multiplication is done by successive addition and shifting right. Division is performed by successive shifting left and subtraction which again involves complementing and adding. It is left to the O.S. to control the above and other such operations.

For solving Scientific and Technical Problems, besides the above Arithmetical operations, Trigonometric and Logarithmic functions have to be used. The computer cannot normally evaluate such functions. So the System Programmers develop certain subprograms called FUNCTIONS to evaluate them by the use of certain Numerical Methods. These Functions are also stored in the ROM in a separate area called the Library area and hence called Library Functions. Whenever a Trigonometric or Logarithmic expression is come across in the user's program, the Processor will search for the Subprogram (Function) corresponding to it in the Library of the Memory, execute the operations in it and return the value of the function.

7.5 SECONDARY MEMORY DEVICES

When a computer is to store a number of programs and the related data, at a time, especially when a single processor is to be operated from a number of terminals, a large memory unit will be necessary. But using a large size core or static memory unit will prove very costly. Since the processor can execute only one instruction at a time, a Secondary Storage Device, which is comparatively cheaper, is used in addition to the Main memory. The size of the secondary memory will be many times that of the Main memory. The data and programs of the various users from the various terminals will be stored in the secondary memory. Whichever program segment that has to be currently executed and the related data are transferred from it to the Main memory. After the processing is over, the program segment and the result are sent to the secondary memory, to give place to the next program segment and the data to be processed. This is essential, since the processor can execute the instructions that are stored in the main memory only. Also, the data transfer rate from the secondary memory is very slow. But provision of secondary memory facilitates processing of programs that require large storage area though using a small main memory. The apparent increase in memory size thus availed is called VIRTUAL MEMORY.

The secondary memory is also called 'Auxiliary Memory' or 'Peripheral Memory' or 'External Memory'. Magnetic Tapes or Cassette Tapes or Floppy Disks or Winchester Disks can be used as Secondary Store/Memory units. But the Floppy or the Winchester Disk is preferred to the tapes. This is because the tapes provide Sequential Access only. That is, the records in them have to be reached one by one in the order of their recording. This will result in different access times for different records, whereas the access time should be uniform for all records in a memory thus device. So the disks are preferred to the tapes since the disks facilitate direct access to any sector, thus providing a uniform access time for any record. The Winchester Disk is used as the secondary device in all the computers. Secondary memory is very useful in storing data-files and data bases also.

Summary

The functions of the Memory unit are—

(i) to store the Operating System
(ii) to store the user's program
(iii) to store the Library Functions
(iv) to store the input data
(v) to assemble the output results
(vi) to store certain data that are to be referred to or retrieved later.

The Memory unit is made up of 2 devices, namely—(i) the Main Memory, and (ii) the Secondary Memory.

The main memory is of 2 sections, namely—(a) the RAM, and (b) the ROM. The ROM is an Electronic memory. It stores the O.S. and the Library Functions. The user can use them but cannot enter his data or program in the ROM, that is the user has only read-access but no write-access to ROM.

RAM provides both the Read and Write accesses. It is to store the users' programs and data. It may be Core type or static (Electronic) type. The processor can execute the instructions stored in the main memory only.

When a processor is operated from a number of terminals, to store all the programs and the related data, a large memory unit will be necessary. Since a large main memory will be costly, a secondary storage unit is to be used. This will be in the form of a Winchester (Hard) Disk unit whose size will be about 10 times that of the main memory. The data and programs from the various terminals are stored in the Secondary memory. Whichever program segment that has to be currently executed and the related data are sent to the main memory. After the execution is over, they are sent back to the Secondary memory to give place to the next program segment to be executed and its related data.

Exercise/Questions

7.1 What are the functions of the Memory?

7.2 Distinguish between—(i) Main memory, and (ii) Secondary memory.

7.3 Differentiate between—(i) RAM, and (ii) ROM.

7.4 Explain why are disks preferred to tapes as secondary storage devices.

7.5 Differentiate between—(i) Direct access, and (ii) Sequential access.

7.6 Differentiate between—(i) Secondary memory, and (ii) Virtual memory.

for NOTES

"This page is Intentionally Left Blank"

Part B

Elements of Programming

Chapter Eight	Introduction to Programming *73 – 82*
Chapter Nine	Constants and Variables in FORTRAN Program *83 – 89*
Chapter Ten	Input-Output Statements *90 – 106*
Chapter Eleven	Arithmetic Statements and Library Functions *107 – 118*
Chapter Twelve	Control Statements *119 – 131*
Chapter Thirteen	Loop Statements and Processor Control Statements *132 – 146*
Chapter Fourteen	Sub-Programs (Sub-routines) and Specification Statements *147 – 165*
Chapter Fifteen	Additional Programming Exercises *166 – 167*

"This page is Intentionally Left Blank"

Chapter Eight

Introduction to Programming

8.1 INTRODUCTION

The DIGITAL COMPUTER has been an ELECTRONIC DATA PROCESSING (EDP) Machine. Of course, its application has now been upgraded to INFORMATION PROCESSING (DATA BASE) and KNOWLEDGE PROCESSING (ARTIFICIAL INTELLIGENCE).

8.2 DEFINITION OF DATA PROCESSING AND DATA

DATA PROCESSING refers to any procedure for receiving information (DATA) and producing a specific result.

DATA is the set of details or information given to the computer. The computer processes the data and gives out the desired result.

8.2.1 Different Forms of Data

In case of data processing by computers, the data is of 3 forms, namely, (i) FILE, (ii) RECORD, and (iii) FIELD.

A 'file' is a collection of records that have similar information (eg,), An opening stock file contains records each of which gives the identification of a material and the opening stock quantity of that material.

As such, a 'RECORD' contains information about a person or a material or one set of readings from an experiment or a survey.

A record contains one or more fields (eg,), In an opening stock record, the material number, name and quantity are the fields.

The computer processes a file taking record by record. In a record, the processing is done on the fields. The fields may be numbers or strings, like names.

8.3 MAJOR STEPS IN DATA PROCESSING

The 3 major steps in Data processing are (i) DATA PREPARATION, (ii) PROGRAM PREPARATION, and (iii) COMPILING AND RUNNING THE PROGRAM.

8.3.1 Data Preparation

The preparation of data is an essential step in data processing since data is to be presented to the computer in a form which it could reckon and be able to manipulate easily to give the result (eg), To solve a quadratic equation, say $4x^2 - 3x + 5 = 0$, the input data should be the values of the coefficients a, b and c (ie), $4, -3$ and 5 and not the equation.

8.3.1.1 Steps in Data Preparation

The following are the steps in the data preparation:

(i) Analysing the system and fixing up the data fields (eg). If, in an AC circuit, it is required to find the power factor, the input data fields are to be decided as the values of VOLTAGE, CURRENT and POWER.

(ii) COLLECTION OF DATA : This is done by performing an experiment or survey and taking readings. Or the data can be obtained from some other organisation or department.

(iii) DESIGNING a standard input format: If processing involves a number of sets of readings, each such set is a record. All the records should be of same form, especially if the processing is to be done using FORTRAN or COBOL. So a standard input format should be designed. This design will involve the following steps:

   (a) Fix up the sequence of fields.

   (b) Fix up the length of each field allowing for maximum possible length. While fixing up the length, an extra column or character position should be allowed each for recording the MINUS sign and decimal point.

   (c) Write down the configuration of the fields. While doing so, allow for minimum one blank between adjacent fields. (This is not necessary in case of processing with FORTRAN, since a FORMAT statement is used in the program defining the nature and length of each field in a record. In case of COBOL also, this is not necessary since the DATA division entries describe the length and nature of each field in the record).

   (d) Draw the configuration on a LAY-OUT FORM.

   (e) Mark the length and nature of each field in the input format.

To indicate the nature of each field, certain codes may be used as indicated below:

CODE	NATURE OF FIELD
A	ALPHABETIC
×	STRING inclusive of blanks
9	Numeric field
.	Decimal point
–	Negative sign

eg,

AAAA	Indicates an alphabetic field of 4 characters (This can be written as 4(A) also)
5X	Indicates blanks or string field of 5 characters
9(5)	Integer number of 5 digits
999.99	A real number with 3 integer digits and 2 decimal positions (6 characters in all inclusive of decimal point)

The Fig. 8.1 will illustrate the steps in design of an INPUT FORMAT.

Column No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Field Name	A								B								C						
Field Description	-999.99							Blank	-999.99							Blank	-999.99						

Fig 8.1 Input Record Format

(iv) RECORDING INPUT DATA

After designing the input format, the input fields/readings in each record/line have to be recorded in multiple column sheets according to the format designed.

While so recording (a) Numeric fields are to be entered right justified (b) string fields are to be entered left justified, and (c) one line should contain only one record or one set of readings and one column one character only.

The input format is used as reference material for recording data in a uniform format in all records as also for writing format statements, if any, in the program.

(v) DATA ENTRY

The data recorded in multi-column is entered on a floppy disk through a key-board or punched on cards with one card per record.

EXAMPLE 8.1 About 100 quadratic equations in the form $Ax^2 + Bx + C = 0$ are to be solved using the computer. The maximum value of any coefficient is 438.87 and minimum value is -284.92. Design the format of the input record.

Solution:

(1) Fix up the input fields. The input will be values of coefficients A, B and C.

(2) Fix up their sequence. The sequence will be A, B and then C.

(3) Fix up the length of each field.

The maximum length will be that of -284.92 (ie) 7 character positions inclusive of the negative sign and the decimal point. So each variable A or B or C can be allocated 7 character length.

(4) Write down their configuration:

CHARACTERS	1–7	A
	8	BLANK
	9–15	B
	16	BLANK
	17–23	C

(5) Draw the configuration on layout form and give the field description (ie) length and nature of each field (Fig. 8.1).

(6) Verification: The data entered is verified line by line/record by record, using the data recorded in multiple column sheet as reference and errors, if any, in transcription are corrected.

(7) Sort the data records and arrange in an order.

8.3.2 Program Preparation

Program is a set of unambiguous instructions given in proper sequence to get a problem solved through the computer. The PROGRAM PREPARATION involves the following steps:

(i) Getting the input format or preparing it, if it is not already there. The input format is essential to name the fields in the INPUT (READ) instruction in the order they occur from left to right in the input record.

(ii) PREPARATION OF OUTPUT FORMAT : This would be essential to write the output (WRITE) instruction as also to get the result displayed in an easily readable and well-presentable form.

The output may be mostly displayed on the screen or got as a PRINT-OUT.

The steps for designing the format of output will be more or less similar to the INPUT-formats. The steps will be (a) Fix up the fields to be displayed; (b) Fix up their sequence; (c) Fix up the length of each field, allowing for maximum possible length; (d) Write down their configuration. Allow a few blanks between adjacent fields to provide better readability. Also center the data to be displayed by allowing more or less equal number of blanks on its either side, and (e) Write down the field descriptions. The following problem will illustrate the above procedure.

EXAMPLE 8.2 Design the output FORMAT for the problem given in 8.1. the output will be on an 80 column screen or an 80 column print-out.

Solution:

(1) Fix up the fields: The fields in the input record, namely, A, B and C, are to be given in the output record also.

Besides them, the roots x_1 and x_2 have to be displayed. In case of imaginary roots, both x_1 and x_2 will have 2 components, namely, 'real part', say X_1R, X_2R and the 'imaginary part', say X_1I and X_2I, respectively. For uniformity's sake, these 4 fields will be provided in all cases. In case of real roots, the imaginary components X_1I and X_2I will be recorded as 0.

(2) Fix up the sequence of fields. The sequence will be $A, B, C, X_1R, X_1I, X_2R, X_2I$.

(3) Fix up length of each field: The lengths of A, B and C will be same as in the input record (ie) 7 characters.

The lengths and nature of X_1R, X_1I, X_2R and X_2I can be assumed to be same as A or B or C.

(4) Write down their configuration allowing for a few blanks between adjacent fields. Also centre the data to be printed.

FIELD DESCRIPTION	LENGTH	CONFIGURATION
BLANKS	7 COLUMNS	COLUMN 1–7
A	7 COLUMNS	COLUMN 8–14
BLANKS	3 COLUMNS	COLUMN 15–17
B	7 COLUMNS	COLUMN 18–24
BLANKS	3 COLUMNS	COLUMN 25–27
C	7 COLUMNS	COLUMN 28–34
BLANKS	3 COLUMNS	COLUMN 35–37
X_1R	7 COLUMNS	COLUMN 38–44
BLANKS	3 COLUMNS	COLUMN 45–47
X_1I	7 COLUMNS	COLUMN 48–54
BLANKS	3 COLUMNS	COLUMN 55–57
X_2R	7 COLUMNS	COLUMN 58–64
BLANKS	3 COLUMNS	COLUMN 65–67
X_2I	7 COLUMNS	COLUMN 68–74
BLANKS	6 COLUMNS	COLUMN 75–80

(5) DRAW the FORMAT in the output layout form and enter the field descriptions (refer Fig. 8.2).

	1 2 3 4 5 6 7	8 9 10 11 12 13 14	15 16 17	18 19 20 21 22 23 24	25 26 27	28 29 30 31 32 33 34	35 36 37
FIELD DESCRIPTION	BLANKS $7x$	A −999.99	BLANKS $3x$	B − 999.99	BLANKS $3x$	C − 999.99	BLANKS $3x$

Contd...

Contd...

38 39 40 41 42 43 44	45 46 47	48 49 50 51 52 53 54	55 56 57	58 59 60 61 62 63 64	65 66 67	68 69 70 71 72 73 74	75 76 77 78 79 80
X_1R −999.99	BLANKS $3x$	X_1I − 999.99	BLANKS $3x$	X_2R − 999.99	BLANKS $3x$	X_2I − 999.99	BLANKS $6x$

Fig 8.2 Output Record Format

(iii) FIX UP THE NUMERICAL METHOD OR THE LOGIC of solution to the problem.

In most cases, the computer method of solution will be different from the methods used in manual solution. For example, the GCD (greatest common divisor) of any 2 integer numbers, say, M and N, has to be computed using the following steps:

Step 1. Check whether M = N. If so, GCD = M = N.

Step 2. Else, if M > N, then make M = M − N and go to Step 1.

Step 3. Else, make N = N − M and go to Step 1.

(eg,)

In case $M = 35, N = 25$, the above steps will be executed on them as follows:

INPUT	$M = 35$	$N = 25$
ITERATION 1	$M = 35 - 25 = 10$	$N = 25$
ITERATION 2	$M = 10$	$N = 25 - 10 = 15$
ITERATION 3	$M = 10$	$N = 15 - 10 = 5$
ITERATION 4	$M = 10 - 5 = 5$	$N = 5$

SINCE $M = N$ now, the iteration will stop here.

$GCD = M = N = 5$.

(iv) WRITE DOWN THE ALGORITHM and/or DRAW THE FLOW-CHART.

An ALGORITHM is an unambiguous ordered sequence of steps in all details, that leads to the solution of a given problem.

The algorithm incorporating the above numerical method will be as follows:

ALGORITHM GCD (Given 2 integer numbers M and N, to find their GCD).

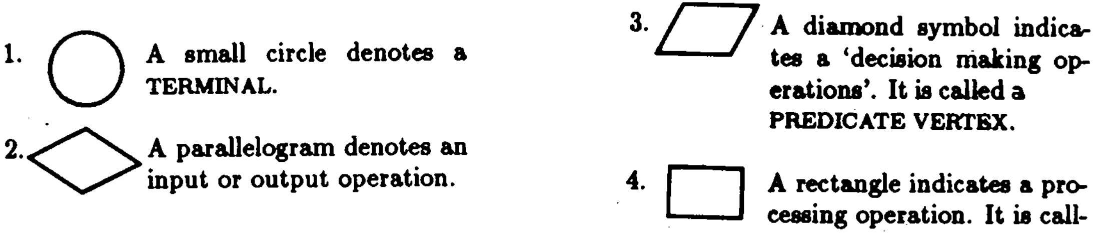
```
Step 1      (Get input data)
            READ (M, N)
Step 2      If $M = N$, then Go To Step 5
Step 3      Else
            If $M > N$, then
              Set $M \leftarrow M - N$
              Go To Step 2
Step 4      Else (cf $N > M$)
              Set $N \leftarrow N - M$
              Go To Step 2
Step 5      Set $GCD \leftarrow M$
Step 6      (output the result)
            WRITE (GCD)
Step 7      (Finished)
            EXIT
```

Statements written within brackets in the algorithm are explanatory comments.

Instead of writing the solution steps in the form of an algorithm, they can be drawn in the form of a FLOW-CHART. A flow-chart is sometimes called a BLOCK-DIAGRAM. It is a graphical form of the algorithm.

A flow-chart consists of a number of BOXES or BLOCKS each of which represents an operation. The sequence of operations are indicated by the directed lines connecting the boxes.

The several kinds of symbols on boxes used in flow-charts are as follows (Fig. 8.3).



1. A small circle denotes a TERMINAL.
2. A parallelogram denotes an input or output operation.
3. A diamond symbol indicates a 'decision making operations'. It is called a PREDICATE VERTEX.
4. A rectangle indicates a processing operation. It is called a FUNCTION VERTEX.

Fig 8.3

The algorithms GCD can be drawn in the form of a flow-chart as follows (Fig. 8.4).

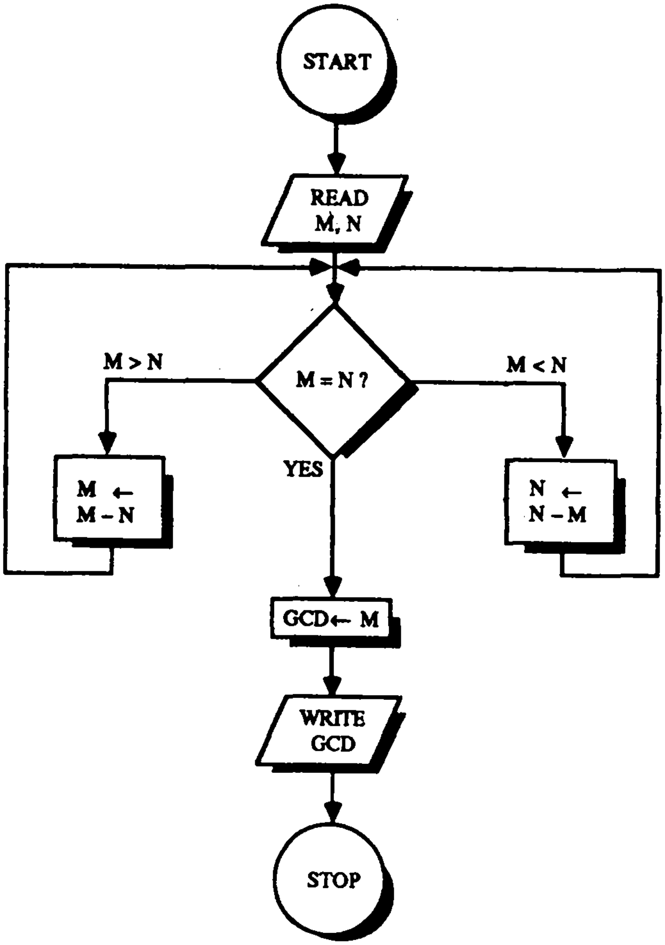


Fig 8.4

The program statements are to be written with the ALGORITHM or FLOW-CHART as reference.

(v) FIX UP THE PROGRAMMING LANGUAGE : The user's programs are written in what are called HIGH LEVEL LANGUAGES. The choice of one among them depends upon the problem to be solved. For scientific and technical problems, FORTRAN is preferred. For business problems, COBOL is preferable. BASIC and PASCAL are capable of processing both technical and business problems.

In this book, the FORTRAN language is discussed.

(vi) CODING THE PROGRAM : When once the appropriate PROGRAMMING LANGUAGE is chosen, the program is written in that language, using INPUT/OUTPUT formats and the ALGORITHM or FLOW-CHART as reference. CODING means writing the detailed procedure to solve a problem, in a computer language.

(vii) ENTERING THE PROGRAM : CODING of the program is done in special coding sheets in case of COBOL and FORTRAN or in any ordinary sheets in case of other languages. It is then entered through the key-board in a floppy disk or punched on cards using the off-line punching machine.

(viii) VERIFYING THE PROGRAM ENTERED : The program entered in an input medium like floppy disk or card will be verified for any transcription errors and errors, if any, are corrected. The transcription errors are—(a) insertion of an extra character, (b) omission of a character, (c) using incorrect character, and (d) transposition of 2 adjacent characters.

The verified and corrected program is called the SOURCE PROGRAM.

8.3.3 Compiling and Running the Program

The computer can operate only on instructions in a coded form which is called MACHINE CODE or MACHINE LANGUAGE. But writing the program in the machine language is very difficult for a user or programmer. A programmer always makes use of an easier language like FORTRAN, PASCAL, BASIC and COBOL to write the program. Such languages are called HIGH-LEVEL LANGUAGES.

A source program in such a high level language is to be translated into a program in the machine language to be executed by the computer. For this, a translator program called COMPILER is to be used.

Using a FORTRAN-COMPILER, which may be on tape or disk, the source program in FORTRAN is translated into Machine code. This process is called COMPILATION. The computer is to be used for COMPILATION (Fig. 8.5). The output program from compilation is called the OBJECT PROGRAM.

The object program may be got on a floppy disk. Besides giving the object program, the computer gives an output on the screen or on the printer, which is called the PRELIST, or the LISTING. The prelist contains the source program, with errors in syntax, if any, being pointed out. (It should be noted here, logical errors, if any, will not be pointed out). The small errors in syntax are called BUGS. These have to be removed. This removal is called DEBUGGING.

Debugging is done on source program and compilation is done again. This may have to be repeated until prelist indicates 'NIL' error.

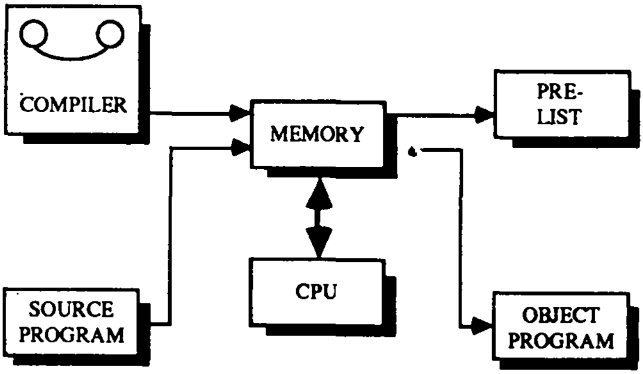


Fig 8.5 Compilation

When once an error-free OBJECT PROGRAM is obtained, it is loaded into the memory. This process of storing a program in the computer memory and making the computer execute it is called the 'STORED PROGRAM CONCEPT.'

If a program is to be run for the first time, it is tested with same hypothetical records or data. This is to check whether the LOGIC in the program is correct and it is called the PROGRAM CHECK-OUT.

If this check gives correct results, the object program is used to process the actual data records. Else, the source program is altered suitably, compiled and tested again.

For running the program with test data or actual data records, the object program is loaded and stored in the memory and the input data is given to the computer. The computer executes the instructions in the program on the input data and displays the result on a screen or prints on a paper (Fig. 8.6).

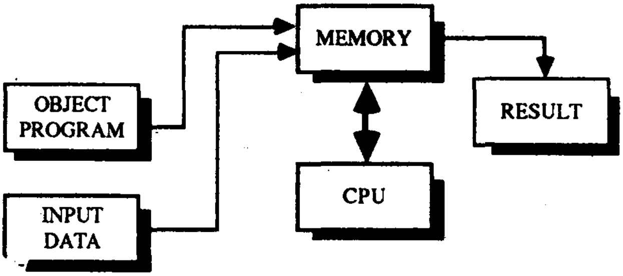


Fig 8.6

Summary

The input details given to the computer are called DATA. For the computer to process the data, instructions are to be given in a language called PROGRAMMING LANGUAGE. The set of instructions is called the PROGRAM. A flow-chart is drawn, to fix up the sequence of logic, before writing the program.

The program written in a high-level language is procedure-oriented.

To be executed by a computer, its instruction should be machine-oriented (ie,) in machine code.

Using a systems program known as COMPILER, the user's program (source program) is converted into a program in machine code (object program). The object program is used to process the data.

Exercise/Questions

8.1 What are the 3 major steps in DATA PROCESSING?

8.2 Describe the various stages of data preparation.

8.3 What are the various steps in preparation of INPUT-FORMAT?

8.4 Differentiate among—(i) Collection of Data, (ii) Recording of Data, and (iii) Data Entry.

8.5 What are the various steps in program preparation?

8.6 Explain the terms—(i) SOURCE PROGRAM, (ii) BUGS, (iii) OBJECT PROGRAM, (iv) COMPILER, (v) ALGORITHM and (vi) FLOW-CHART.

8.7 The calculation of the power factor and power factor angle in an A.C. circuit is to be computerised. Assuming that the voltage in the circuit is around 230 and the maximum value of current is about 15 amps and readings of voltage, current and power can be taken to the accuracy of 2 decimal places, design the format of INPUT RECORD.

Assuming the p.f. is to be given to the accuracy of 5 decimal positions and p.f. angle to the accuracy of 3 decimal places, design the format of the output to be displayed on an 80 character screen.

8.8 Write an ALGORITHMIC solution for finding the roots of a quadratic equation.

8.9 Draw the flow-chart, given an integer number, to find its factorial.

for NOTES

Chapter Nine

Constants and Variables in FORTRAN Program

9.1 FORTRAN LANGUAGE

FORTRAN is a shortened form of FORMULA TRANSLATION. It is a high level programming language. It is used in writing programs for solving scientific and technical problems. For example, the formula/equation, $S = ut + \frac{1}{2}at^2$, which we come across in dynamics, has to be translated as

$$S = U * T + (A * T * *2)/2.0$$

when it is used in a program. The FORTRAN language gives the rules for such translations.

In this book, the version of FORTRAN known as FORTRAN IV is discussed in detail.

9.2 FORTRAN PROGRAM

A FORTRAN program is made up of a series of statements. These are written in 80 column coding sheets (refer Fig. 9.1). Only one character is to be recorded in each column in the sheet. Columns 1–5 may be used to write numbers by which the statement in that line may be subsequently referred. (There is no necessity that every statement is to be numbered.) A statement number may be any unsigned integer number within 32768. The statement numbers can be assigned in any order since the sequence of operation depends only on the order of statements and not on their statement numbers.

COMMENTS are often written at the beginning of the program giving a brief description of it. Comments may also be written in the middle of the program to explain any complicated logic involved in a statement. Such comments may be written in columns 2 to 72 of a line with a 'C' in column 1 of that line. These comments are not compiled, but are printed in the prelist (listing).

FORTRAN statements are written one in a line in columns 7 to 72. A line can contain a single statement only. But a statement may extend to maximum 10 lines. The column 6 is used to record a mark of continuation. The first line may have in its column 6, a blank or a 0 recorded in it. The second and subsequent lines have some non-zero character recorded in column 6. In practice, nos. 1 to 9 are recorded in these lines to indicate that they are continuation lines 1, 2 etc... 9, respectively.

While writing FORTRAN statements in columns 7 to 72, blanks may be used between terms to provide better readability. (Blanks in these columns will be ignored by the compiler). Also, the statement may begin from column 7 or in any subsequent column. Columns 73 to 80 may be used for recording the identification of the programmer and/or the program. These columns are ignored by the compiler.

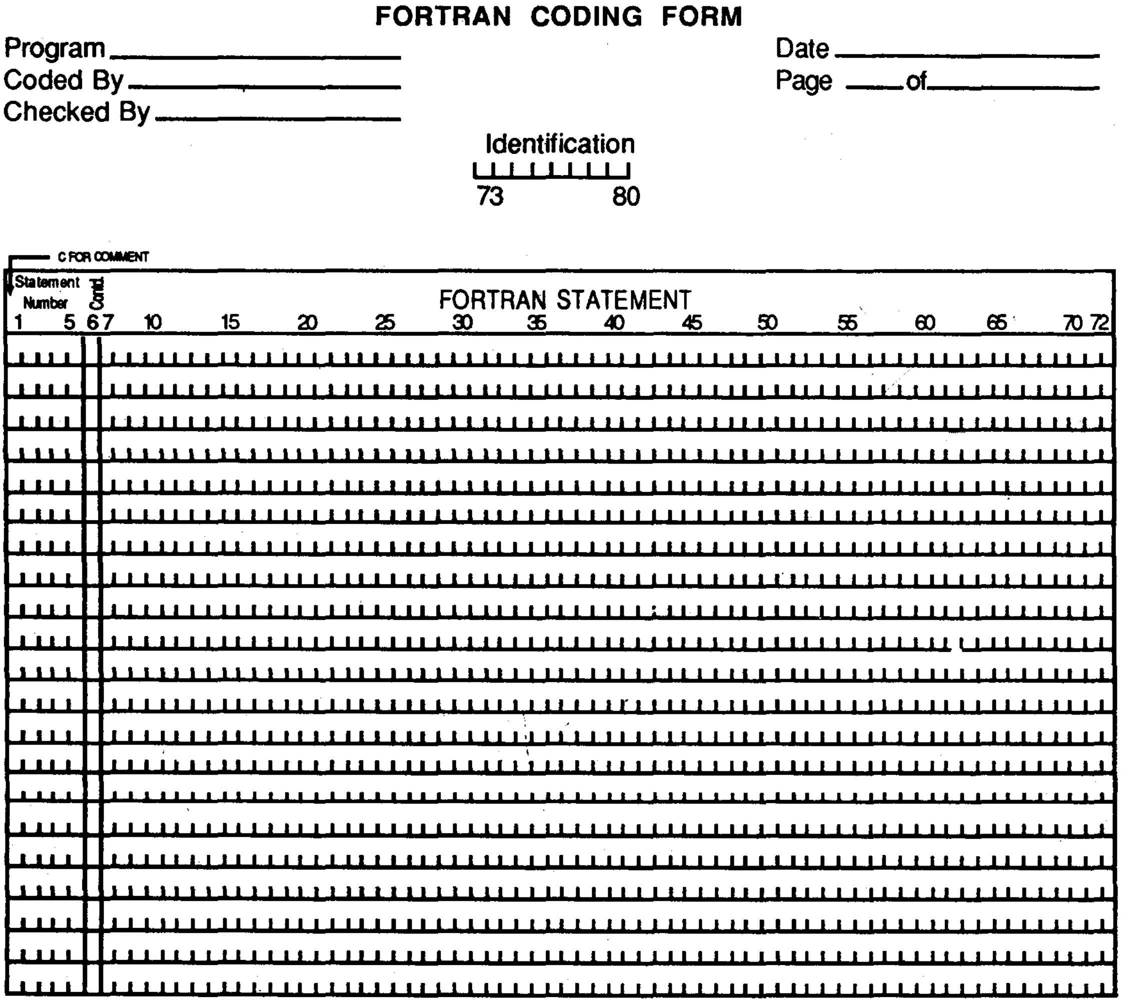

FORTRAN CODING FORM
Program
Date
Coded By
Page of
Checked By
Identification
73
80
C FOR COMMENT
Statement Number
Cont
FORTRAN STATEMENT
1
5
6
7
10
15
20
25
30
35
40
45
50
55
60
65
70
72

Fig 9.1 FORTRAN Coding Form

9.3 ELEMENTS OF FORTRAN LANGUAGE

The basic elements of FORTRAN are—(i) CONSTANTS, (ii) VARIABLES, (iii) I/O statements, (iv) ARITHMETIC statements, (v) Control Statements, (vi) Sub-routine Statements, and (vii) Specification statements (eg) TYPE, DIMENSION, COMMON, EQUIVALENCE Statements.

9.4 CONSTANTS AND VARIABLES

The quantities in a program can be specified in two ways, namely—(i) CONSTANTS, and (ii) VARIABLES.

For example, in the statement AVERAGE = TOTAL/IO, AVERAGE, and TOTAL are variables and 10 is a constant.

9.4.1. Constants and Their Types

CONSTANT is a number that is used to specify a quantity. It is a known quantity when used in an Arithmetic expression (eg) $A = \pi R^2$ where A = area of circle, and R = radius, and π is a constant of value 3.141592.

In a FORTRAN expression, any constant is to be represented by the number itself. The statement to compute the area of a circle in terms of its radius R should be written as

$$A = 3.141592 * R * R.$$

There are two types of constants that may be used in FORTRAN, namely—(i) INTEGER CONSTANTS, and (ii) REAL CONSTANTS.

9.4.1.1 Integer Constants

These were previously called FIXED POINT constants. An integer constant is a number without a decimal point. It may be positive or negative. If negative, it is preceded by the minus sign '–'. An unsigned constant is always taken as positive. Examples of INTEGER CONSTANTS are 0, +92, 180, –68 etc.

The integer constants used in FORTRAN must not have values below –2147483647 or above 2147483647.

9.4.1.2 REAL Constants

These were previously called FLOATING POINT CONSTANTS.

A REAL constant is an INTEGER written with a decimal point and a decimal fraction; or a mixed integer and decimal fraction. It may be positive or negative (eg,) 365., 365.0, –368., –18.0, 0.2, .28, –.36, 46.87, –87.46 etc.

The minimum value of a real constant used in FORTRAN should be 16^{-63} (approximately 10^{-75}) and the maximum value 16^{63}, (approximately 10^{75}).

COMMAS should not be used to punctuate either INTEGER or REAL constants in FORTRAN (eg) 1,000 or 15,450.08 are not valid.

9.4.1.2.1 Forms of REAL Constants

The maximum number of significant digits in the real mode of operation of FORTRAN is 7 only. As such, a REAL CONSTANT of greater digits is expressed in E notation.

(eg,)

3.56E8	or 3.56E + 8, or
–3.56E – 7	or –3.56E + 7, or
(+ sign is optional)	3.56E – 8 etc.

3.56E8 stands for 3.56×10^8 E notation is used to represent very small numbers also (eg,) 0.0000045 can be represented as 4.5E − 6 which is equivalent to 4.5×10^{-6} in ordinary notation. This method of representing large or very small numbers is called E NOTATION.

Any REAL constant which is of more than 7 digits should be written with E notation (ie) in the Exponential form. Even then, the accuracy of the results will be up to 7 significant digits only. In case higher accuracy is required, the numbers are to be operted in the DOUBLE PRECISION mode which gives the result up to 16 significant digits. Double precision mode can operate only on REAL numbers which are to be expressed in a special type of Exponential form called DOUBLE PRECISION FORM. In this form, D is used in place of E (eg) 4.8367D12 or 4.876D − 11.

As such, the real constants can be in any of the 3 forms, namely (a) DECIMAL form (eg) 24.5, −28.5, 17.0, −17.0, .05, −.84 etc. (b) Exponential form (eg) 4.8E9, 4.67E − 8, −4.54E7, −4.72E − 6 etc. (c) DOUBLE precision form (eg) 3.62 D10, −3.26 D11, 3.26 D − 8, −3.79D − 7 etc. In both E and D notations, the value of exponent should not be below −75 and above 75.

9.4.2 Variables

As stated earlier, the quantities in a FORTRAN PROGRAM may be represented as CONSTANTS or VARIABLES.

A VARIABLE is defined as a symbol which represents a quantity which is taken through an input medium or whose value is specified by some other statement. (eg) In the statement B = 2.5 × A, both B and A represent the VARIABLES. A and B are written in place of the quantities they represent and hence are called VARIABLE NAMES.

Just as contants, variables are also of 2 types, namely (1) INTEGER or FIXED POINT VARIABLES, and (2) REAL or FLOATING POINT VARIABLES.

9.4.2.1 Naming Rules for Variables

FORTRAN has rules for naming the variables. They are as follows:

(i) Any name may have 1 to 6 characters (This excludes subscripts, if any).

(ii) The first character must be an alphabet.

(iii) The name of an INTEGER VARIABLE should begin with any one of the 6 letters, I, J, K, L, M or N.

(iv) The name of a REAL VARIABLE begins with any alphabet except the above six.

(v) The second and successive characters may each be an alphabet or a digit, but cannot be any special character. (Refer next chapter for a list of special characters).

(vi) No embedded blanks are permitted within the name.

(vii) The name should not be a RESERVE WORD.

(Reserve words are those that convey specific meaning to the compiler. The words READ, WRITE, GO TO, IF, DO, DIMENSION CONTINUE, COMMON, EQUIVALENCE etc are all RESERVE WORDS.

The following are examples of valid names for INTEGER variables: J, ITEM, NFACT, MTLNO, L1, K12, KEY.

The following are a few examples of valid names for REAL VARIABLES. A, BRDTH, COST, DUE, RATE, VALUE, X, Y12, etc.

9.4.2.2 Explicit method of naming

The method discussed above for naming variables is called IMPLICIT SPECIFICATION or method. This method has certain limitations. For example, if the user wants the names MASS or LENGTH to be REAL, he has to change them to say, AMASS and ALNGTH, respectively. He may have to write as FORCE = AMASS * ACCN instead of FORCE = MASS * ACCN.

To overcome this problem an EXPLICIT SPECIFICATION or method of naming is provided for. The explicit method of naming is done by TYPE statements.

A TYPE statement is a specification statement and not an executable statement. The form of the TYPE statement to define explicitly one or more variable names as integer is the word INTEGER. (eg) INTEGER EMPNO, SENO, GCD, HCF. Similarly, the form of the Type statement to define explicitly one or more variables as real, is the word REAL (eg) REAL MASS, LNGTH, MAX, MIN.

FORTRAN provides one more type statement, namely DOUBLE PRECISION.

As discussed earlier, the numbers to be processed in the DOUBLE PRECISION mode should be in the double-precision form (Art 9.4.1.2.1).

Besides this, the names of variables to be operated in double precision mode have to be declared as such, by a type statement, for example:

DOUBLE PRECISION AMINOR, AREA, DIA.

The Type statements, if any, in the program, should precede all the executable statements. The mode of a variable defined by an EXPLICIT definition cannot be changed in the middle of the program.

9.4.3 Subscripted Variables

Subscripted variables are elements in ARRAYS.

An ARRAY is a group of similar elements (eg) an ARRAY of NATURAL numbers.

The arrays may be single dimensioned or multi-dimensioned. In practice, we come across mostly single dimensioned arrays and/or two dimensioned arrays.

The single dimensioned arrays are called VECTORS and the two dimensioned arrays MATRICES.

A single name can be used to represent the numbers or elements in a Vector or Array. In such case, provisions should be made to refer to a specific member or element in that vector or array. This is accomplished by the use of SUBSCRIPTS.

A SUBSCRIPT consists of a constant or variable or limited expressions attached to a variable name and contained within parentheses. In case of a vector, a single subscript is used with the variable name (eg) A(1), A(2), A(3) ... ... A(100).

The subscript must be an INTEGER constant or INTEGER VARIABLE (eg) x(20), y(I). It cannot be 0 (ZERO).

In case it is a variable, its value should be defined elsewhere in the program or evaluated within the subscript expressions.

(eg) G(2 × I)

A subscripted variable may be in the INTEGER mode or REAL mode but the subscript should always be an INTEGER. Its value should not exceed $2^{15} - 1$(32767).

9.4.3.1 Two-Dimensioned Arrays

The elements in a two-dimensioned array or matrix may be identified by the use of double subscripts. (eg) A (4,8). The first subscript is called ROW INDEX and the second COLUMN INDEX.

A (4,8) means it is an element at the junction of row 4 and column 8 in the matrix A.

9.4.3.2 Dimension Statement

Use of subscripted variables facilitates use of LOOP statements in the program which reduce the program-size. When vectors and arrays are used in the program, one or more DIMENSION statements have to be used.

The DIMENSION statement specifies the size of arrays. Size or length of an array means the number of elements or members in it. If a vector (single dimensioned array) contains 100 numbers, its size is 100.

The Dimension statement consists of the word DIMENSION followed by any number of array-size specifications, separated by COMMAS (eg,)

```
DIMENSION A(100),x(150),y(200).
```

Every variable that is used with one or more subscripts must appear in the DIMENSION statement (eg)

```
DIMENSION A(120),B(10,15),C(20,20),X(100).
```

In the above, A and x represent VECTORS, and B and C, MATRICES.

The DIMENSION statement should precede all the executable statements in the program.

If there is a Type statement also, the DIMENSION statement should succeed it.

(eg)

```
REAL NMBR, ITEM
DIMENSION NMBR (12, 12), ITEM (100).
```

Summary

FORTRAN programs are written in a special coding sheet of 80 columns.

CONSTANTS and VARIABLES are used in FORTRAN programs. Both are classified as (1) INTEGERS or FIXED POINT numbers, and (2) REAL numbers or FLOATING POINT numbers. REAL constants are of 3 forms, namely (a) DECIMAL FORM (b) EXPONENTIAL FORM, and (c) DOUBLE PRECISION FORM. Variable are represented by symbols or names. There are 2 methods of naming variables, namely (1) IMPLICIT method, and (2) EXPLICIT method. In the implicit method, INTEGER Variables are assigned names starting with the letters I, or J, or K, or L, or M, or N. Any one of the other 20 letters will be used as the first character in the names of REAL variables.

In the EXPLICIT definition, TYPE statements are used to indicate whether a variable is in INTEGER mode or REAL mode or DOUBLE PRECISION mode.

In case of ARRAYS, subscripted variables are used to specify the element in the array. A DIMENSION statement should be provided in the beginning of the program giving the SIZE of the array.

Exercise/Questions

9.1 What are the basic elements in FORTRAN program?

9.2 Illustrate the two types of constants with examples.

9.3 State whether the following are valid INTEGER constants:

0, +9, −12, 3.2, −6.8, 2258483600.

9.4 State whether the following are valid REAL constants: 48, −63, 63., −48., .82, −.735, 0.732, −0.372, 62.83, 63.0, −42.84, 4528474.0, 524849.78, 4.62E10, −4.26E9, −5.26E − 11, 6.267E7, 6.28D7, 6.82D − 11, −6.44E − 7, 7.24E − 12.

9.5 Differentiate between (a) IMPLICIT, and (b) EXPLICIT definition of variables.

9.6 Check whether the following names are admissible for naming the variable.

A, A1, B21, BRDTH, LENGTH, AREA, N-EMP, MRK +, IN REC, TTL *.

9.7 Identify the mode of the variables below:

YSUB1, JOHN, KOUNT, CTR, NAC, HEIGHT, LCM, NFACL.

9.8 Write short notes on: (i) VECTOR, (ii) Two dimensioned array, and (iii) DIMENSION statement.

for NOTES

Chapter Ten

Input-Output Statements

10.1 CHARACTER SET

FORTRAN program is made up of statements. The statements contain terms that are made up of characters given below. This set of characters is called CHARACTER SET of FORTRAN. They are of 3 types, namely, (1) Letters or Alphabets, (2) Digits, and (3) Special Characters.

LETTERS : A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

DIGITS : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

SPECIAL CHARACTERS :

.	DECIMAL POINT
)	RIGHT PARENTHESIS
(	LEFT PARENTHESIS
+	PLUS
−	MINUS
*	MULTIPLICATION OPERATOR
/	DIVISION OPERATOR
=	ASSIGNMENT SYMBOL
,	COMMA
‘	QUOTATION MARK

(Blank is also treated as a special character)

10.2 TYPES OF FORTRAN STATEMENTS

FORTRAN STATEMENTS can be broadly classified as—(a) I/O statements, and (b) PROCESSING statements.

10.3 I/O STATEMENTS

I/O Statements mean INPUT-OUTPUT statements. These are classified as—(a) INPUT statements, (b) OUTPUT statements, and (c) FORMAT statements. Of these, Input and Output statements are executable. The FORMAT statement is a SPECIFICATION statement.

10.3.1 Input Statements

The digital computer can accept data from cards, tapes, disks or key-board. This process is called INPUT operation. The instruction in the FORTRAN program for an input operation is a statement that contains the verb READ.

The READ statement in FORTRAN IV is structured as follows—

The word 'READ', a set of parentheses which enclose a reference to an input device, a statement number and list of variables which are separated by commas. The reference to input device and the statement number are also to be separated by a comma. (eg)

```
READ (5,10) ALNGTH, BRDTH, HEIGHT
```

The above READ statement means take the values of ALNGTH, BRDTH and HEIGHT, whose format is specified in statement of number 10 from the input device denoted by number 5. The data fields ALNGTH, BRDTH, HEIGHT are mentioned in the sequence they occur from left to right in the input record.

10.3.2 Device Reference Numbers

Conventionally, the following numbers are used to indicate various I/O devices

I/O Device	Device No.
SCREEN/KEY-BOARD	0
TAPE UNITS	1,2,3,4
CARD READER	5
PRINTER	6
CARD PUNCH	7
DISK UNITS	8,9,10,11,12,13.

Note that the input device reference number is an unsigned integer.

10.3.3 Variables in the READ Statement

The list of variables in the READ statement may contain any number of variables. These variables may be of any mode. Also, both integer and real variables can be specified in the same list. (eg)

```
READ (0,10) NMBR, PAY, DEDN, GROSS.
```

10.3.4 Format Statements

As stated earlier, the FORMAT STATEMENT is a SPECIFICATION statement. It is referred to by an INPUT or OUTPUT statement.

Though it can be written anywhere before the END statement in the program, it is advisable to write it immediately below the related input (READ) or output (WRITE) statement. (eg)

Since the FORMAT statement is to be referred by an I/O statement, every FORMAT statement should have a statement number. This number is mentioned in the related I/O statement. (eg)

```
   || READ (5,70) A,B,C,N.
```

In this statement 70 is the FORMAT statement-number. It directs the processor to refer to the FORMAT statement which is numbered as 70 for the information of 'Record Lay-out' that contains the values of the variables A,B,C and N.

The FORMAT statement is constructed as follows—

A statement number, the word FORMAT, and a pair of parentheses containing the information about the record layout. (eg)

```
70 || FORMAT (record layout)
```

The record layout information gives the nature and width of the fields in the record in the sequence they occur in the input or output record. The nature of the field is specified by an alphabet and the width by an integer number following it. (eg) I4 indicates an INTEGER field of 4 columns or characters.

The alphabets indicating the nature of the fields are called codes (eg,) I-code, F-code, X-code, T-code, E-code etc.

The list of FORMAT codes and the type of field each indicates is listed below—

CODE	TYPE OF FIELD
I	Integer number
F	REAL number in the DECIMAL form
E	REAL number in the exponential form (for single precision processing)
D	REAL number in the double precision form (for double precision processing)
A	ALPHABETIC field
X	BLANKS
T	Spacing
H	ALPHAMERIC (String) field to be composed by the processor to be given as output.

10.3.4.1 I-Code

I-Code is used to read numbers in the integer mode (eg,)

```
   || READ (0,20) NMBR,MRK,KOUNT
20 || FORMAT (I4,I3,I5).
```

The above FORMAT statement tells the compiler that, in the input record, the first 4 columns (ie, 1–4) contain the value of integer variable NMBR, the next 3 columns (ie, 5–7) contain the value of the integer variable MRK and the next 5 columns (ie, 8–12) contain the value of the integer variable KOUNT.

The maximum width of an integer is limited to 10 columns only. (ie) I10 is the maximum specification.

10.3.4.2 F-Code

The F-Code is used to read real numbers in the DECIMAL FORM. This specification consists of letter F followed by two integer constants separated by a decimal point (eg,) F6.2. The first integer constant specifies the total field-width and the second the number of digits that appear after the decimal point.

If the data in a field is of 'picture' 999.99, then its description in F-Code will be F6.2.

The maximum field width for F type specification is 28 columns in all, out of which the number of digits after the decimal should not exceed 8 columns.

EXAMPLE 10.1 A card record contains values of 3 variables, x, y and z, punched as follows—

COLUMNS	FIELD IDENTIFICATION	FIELD DESCRIPTION
1–8	X	9999.999
9–15	Y	−999.99
16–24	Z	−9999.999

Write down the READ and the Associated FORMAT statement.

Solution:

```
     READ (5,10) x,y,z
10   FORMAT (F8.3,F7.2,F9.3)
```

It should be noted that the total field width is inclusive of the column holding the negative sign, if any. This applies to I-Code, E-code and D-Code specifications also.

10.3.4.3 E-code

E-code is used in case of real data recorded in the exponential form, (ie) using E notation. In the E-code or E type specification, the control character E is followed by 2 integer constants separated by a point. The first integer constant denotes the total field-width and the second the number of digits that would appear after the decimal point. The field width in this case includes a space for the letter E as also for the sign of the exponent and sign and decimal point for the number.

The following example will illustrate the use of E-code:

EXAMPLE 10.2 The following are the details of actual data recorded in a card. Write a READ statement and associated FORMAT statement—

COLUMNS 1–7	2.867E8
8–10	17.248E11
17–24	-5.46E09
25–31	6.28E – 8
32–39	7.42E – 07
40–47	6.26E – 12
48–54	-6.44E7
55–62	-6.44E – 6
63–80	Blanks

Solution:

```
     READ (5,30) A,B,C,D,E,F,G,H
30   FORMAT (E7.3,E9.3,E8.2,E7.2,E8.2,E8.2,E7.2,E8.2)
```

10.3.4.4 D-Code

The usage of D-Code is the same as E-code except that it indicates to the compiler and processor that the corresponding data has to be processed in the double precision mode, (ie) to the accuracy of 16 digits.

10.3.4.5 A-Code

A-Code is used in case of Alphameric (STRING) data that is to be processed. This code has a limitation in FORTRAN that the field-width should not exceed 4 characters (5 in certain versions of FORTRAN). If the actual field width is 12 characters, it has to be split into 3 sub-fields, each of 4 characters and the READ and FORMAT statements may be—

```
        READ (0,5) B1,B2,B3
5       FORMAT (A4,A4,A4)
```

10.3.4.6 Typical Input Record

A typical input record may contain string data and numeric data that may have quantities in the integer mode, in the real mode, in the decimal form or in the exponential form. The FORMAT statement provides for different codes in the same statement. (eg)

```
        READ (2,10) NEMP, NAME, PAY, PF
10      FORMAT (I6,A4,F7.2,E8.3)
```

In the above record, NEMP is an integer field, NAME a string field, PAY a real number in the decimal form and PF a real number in the exponential form. (The FORMAT statement is written using the INPUT FORMAT as reference).

10.3.4.7 X-Code

The X-Code has 2 applications, namely (a) to specify the blanks, if any, in the record, and (b) to make the processor skip a few characters or fields.

The X-Code is used to specify blanks before the first field and blanks, if any, in between fields. The X-Code is written with the field width followed by the character X. (eg) 10X. This is different from other specifications (eg) I6.

The following example will illustrate the use of X-Code to specify blank-fields:

EXAMPLE 10.3 The format of a card record is—

COLUMN 1–5	BLANKS
6–10	MATERIAL NUMBER
11–12	BLANKS
13–16	MATERIAL IDENTIFICATION
17–18	BLANKS
19–22	QUANTITY
23–24	BLANKS
25–30	RATE IN Rs. and Ps.
31–80	BLANKS

Write the READ and FORMAT statement to read the card.

Solution :

```
        READ (5,15) MNMBR, MNAME, NQTY, RATE
15      FORMAT (5X, I5, 2X, A4, 2X, I4, 2X, F6.2)
```

It should be noted that the blank field to the left of first data field and in between adjacent fields are not named in the READ statement, but specified in the FORMAT statement. The blanks to the right of the last field is neither named in the READ statement nor specified in the FORMAT statement.

10.3.4.7.1 Use of X-Code to Skip Fields

Occasionally, processing may involve only 2 or 3 fields in a record. In such a case, X-code may be used to skip the fields that are not required, while reading. The Example 10.4 will illustrate this.

EXAMPLE 10.4 The records in a tape file are of the format—

Characters 1–5	EMPLOYEE NO.
6–9	EMPLOYEE IDENTIFICATION
10–13	QUALIFICATION
14–17	DESIGNATION
18–21	ORGANISATION
22–28	SALARY

Write a READ and FORMAT statement to input the fields 'Qualifications' and 'Salary' only.

Solution:

```
   || READ (4,25) QUALN, SALARY
25 || FORMAT (9X, A4, 8X, F7.2)
```

10.3.4.8 Repeated Specifications

In case adjacent fields or adjacent groups of fields are of same specifications, an integer can be used before the code character I or F or E etc. to indicate the repeated occurrence. (eg) If the 3 variables A, B, C in a record are each of format F6.2, the READ and FORMAT statement will be as follows—

```
  || READ (0,5), A, B, C
5 || FORMAT (3 F6.2)
```

The following examples illustrate further the 'Repeated specifications'.

EXAMPLE 10.5 A Disk-record is of following format—

Character 1–5	EMPLOYEE NUMBER
6–12	PAY IN Rs and Ps
13–17	BANK A/C No.
18–24	CREDIT IN Rs. and Ps.

Write the READ and FORMAT statement.

Solution:

```
   || READ (10,20), NEMP, PAY, NACNT, CREDIT
20 || FORMAT (2 (I5, F7.2))
```

EXAMPLE 10.6 The layout of a card record is—

COLUMNS 1–5	BLANKS
6–10	IDN NO
11–13	BLANKS
14–19	WAGES IN Rs and Ps
20–22	BLANKS
23–28	OT IN Rs and Ps
29–31	BLANKS
32–36	PF DEDN IN Rs and Ps
37–80	BLANKS

Give the READ and FORMAT statement.

Solution:

```
   || READ (5,10) IDN, WAGE, OT, PF
10 || FORMAT (5X, I5, 2 (5X, F6.2), 5X, F5.2)
```

10.3.5 Reading of Arrays/Vectors

Reading of arrays can be done by the use of Do loops. But FORTRAN has a facility called SELF-INDEXING technique to read the elements of an array or vector. In this type of READ statement, the variables with their variable subscripts are followed by an INDEX DEFINITION (eg)

```
READ (5, 1), (A (I), I = 1, 10)
```

This statement, though executed once, will cause the reading of 10 numbers, namely A(1), A(2), A(3) ... etc... A(10).

≡ **EXAMPLE 10.7** 15 integer numbers each of 4 digits are punched in columns 1 to 60 on a card. Write the READ and FORMAT statement—

Solution:

```
   || READ (5,10) (N (I), I=1,15)
10 || FORMAT (15 I4)
```

It should be noted that the FORMAT statement gives a single CARD/RECORD image.

In many cases, all the elements in a vector or array cannot be accommodated in a single record. Even then, the FORMAT of a single record alone is to be specified. The processor will read the records one by one till the field of maximum index is read. For example, if there are 100 real numbers with 10 in each record each of format F8.3 the READ and FORMAT statements will be—

```
   || READ (2,10) (A(J),J=1,100)
10 || FORMAT (10 F8.3)
```

In this case, the processor will read the 100 numbers from the 10 records at a stretch.

It is not necessary that the starting index should be 1 and/or the final index is to be of the last record.

Also, the index incrementation can be more than one (in which case it should be specified in the READ instruction)

Whatever be the case, the FORMAT statement should be the FORMAT of a single record. Refer to the examples 10.8 and 10.9.

≡ **EXAMPLE 10.8** 72 integer numbers N(1) to N(72) each of 4 digits are recorded on 4 cards, last card containing 12 values only. Write the READ and FORMAT statements to read the numbers starting from N(8) to N(68).

Solution:

```
   || READ (5,10), (N(k),k=8,68)
10 || FORMAT (20I4)
```

≡ **EXAMPLE 10.9** In the above example if alternate values say N(1), N(3), ... etc. ... N(71) are to be read, write the READ and FORMAT statements.

```
   || READ (5,20), (N(I),I=1,71,2)
20 || FORMAT (20 I4)
```

The term I = 1, 71, 2 is called INDEX DEFINITION. The starting or lower index is 1, the final or upper index is 71 and index incrementation is 2. If the index incrementation is 1, it need not be specified (refer Example 10.8).

10.3.5.1 Reading 2 or More Vectors

In certain cases, the input may consist of 2 or more vectors. If there are 2 vectors there are 2 methods of recording them.

Method 1. The elements of the 2 vectors are recorded in pairs, say as A(1), B(1), A(2), B(2), A(3), B(3) etc. In this case, the 2 variables are under the control of the same index. The READ statement in this case will be—

READ (5, 15) (A(I), B(I), I = 1, 50)

Method 2. The elements of the 2 vectors are recorded in 2 separate blocks, say as A(1), A(2), ... etc... A(n), B(1), B(2), B(3), ... etc... B(m). The READ statement in this case will be

READ (5, 20) (A (I), I = 1, 50) (B (J), J = 1, 40) when arrays are to be read DIMENSION statement has to be used. (9.4.3.2)

10.3.6 Reading Records with Different Images/Layouts

As stated earlier, the FORMAT statement gives the LAYOUT of a single record. FORTRAN has provision to read records of different images or layouts, say alternate records of same format. This is accomplished by the use of the character "/" (SLASH) in the FORMAT statement (SLASH in Arithmetic expression indicates DIVISION).

For example, the statement,

FORMAT (6 F6.4/10 F5.2) describes 2 different record layouts, namely 6 F6.4 and 10 F5.2. This indicates the first, third, fifth etc. records are of format 6 F6.4 and the second, fourth, sixth etc. records are of format 10 F5.2. Similarly, the statement.

FORMAT (4I5/3 E10.4/5 F6.2) contains 3 different record layouts.

EXAMPLE 10.10 The values of I and A are punched in alternate cards and B and J in the other alternate cards. I is an integer of 5 digits and J of 4 digits. A is of the format F6.4 and B F7.3. Write a READ and FORMAT statement.

Solution:

```
        READ (5,10) I,A,B,J
10      FORMAT (I5, F6.4/F7.3, I4)
```

Note that the SLASH is provided in the FORMAT statement (and not in the READ statement) to indicate values of I and A are available in alternate records and of B and J with other alternate records.

A very useful application of the SLASH in FORMAT statements provides for the reading of a single record according to the first layout in the FORMAT, and then reading several records according to the second record layout. For example, if the final index N of an array is specified only at the time of execution of the program, the statements will be as

```
        READ (5,10)N,(A(I),I=1,N)
10      FORMAT (I3/(10 F8.3))
```

This means the first card contains the value of N in the format I3 and the second and successive card contains the elements of the vector A. 10 elements per card, each element of the form F8.3: Note that in this case there should be an extra pair of parentheses called INTERIOR PARENTHESES surrounding the second layout specification [(10 F8.3)].

The READ and FORMAT statement in Examples 13.6 and 14.5 also illustrate this.

10.4 OUTPUT STATEMENT

The output statement contains the word WRITE whatever be the output device, be it card-punch, tape or disk or print.

The use of WRITE statement is very similar to the use of READ statement. The general form of an output statement will be the identifying word WRITE followed by a device identification number, and a FORMAT statement number within parentheses, and then a list of variables whose values are to be recorded (displayed or printed). The related FORMAT statement has the same function as in the case of READ statement. It specifies the layout of an output record. In the case of a screen or print-out, each line is a record. It is written using output format as reference.

EXAMPLE 10.11 Write an output statement and the related FORMAT statement to record on Tape the following data—

Character 1–5	PART NO.
6–9	PART NAME
10–12	STOCK QUANTITY
13–18	RATE IN Rs and Ps

```
   ||  WRITE (3,10) NPART, NAME, NQTY, RATE
10 ||  FORMAT (I6, A4, I3, F6.2)
```

10.4.1 Output Display on Screen and Printer

The fields in a record displayed on a screen or print-out can be made to provide neat appearance and good readability by allowing blank spaces in-between adjacent fields as also by centering the record displayed. The X-code or X-conversion can be effectively used for spacing the output fields. When used in the FORMAT statement associated with WRITE statement, it causes a number of blanks to be inserted—(eg)

```
   ||  WRITE (0,10) K,X,Y
10 ||  FORMAT (27X, IJ, 4X, F7.2, 4X, F6.2)
```

The above statements will cause a space of 27 blanks before the data field K and 4 blanks each between the fields K and X and X and Y.

This way of providing leading blanks and blanks in-between fields is called HORIZONTAL SPACING.

10.4.1.1 H-Code

An important feature of the WRITE statement is that it can composed CHARACTER STRINGS and print or display them. These strings may be composed of the characters given in the CHARACTER-SET. These need not be given in any input record. (eg) The title 'AN INTRODUCTION TO FORTRAN' can be composed and displayed by the following instruction

```
   ||  WRITE (0,50)
50 ||  FORMAT (27X, 26HAN b̸ INTRODUCTION b̸ TO b̸ FORTRAN)
```

Please note b̸ indicates blank.

The data composed by the computer is called LITERAL DATA. The literal data will not have corresponding variable in the output list of the WRITE statement. In fact, the literal data has to be actually given in the FORMAT statement using an H-code (HOLLERITH CODE).

The H type specification consists of the code letter H, preceded by an integer which gives the length of literal data including embedded blanks. The letter is immediately followed by the literal data to be displayed.

In the above example, the literal data to be displayed is ‘AN INTRODUCTION TO FORTRAN’ which immediately follows the code H. The integer number 26 before H indicates that the above literal data is of 26 characters inclusive of embedded blanks. The specification of 27X in the FORMAT is to centre the literal data in a screen of 80 characters.

10.4.1.2 Alternative Method to H-Code

The use of H-Code is not handy, since the number of characters in the literal data has to be specified with it. Any error in counting the number of characters inclusive of blanks may lead to ‘bugs’. So the alternative method of using SINGLE QUOTES is adopted. In this method, the literal data is enclosed within QUOTES within FORMAT statement as illustrated below

```
      WRITE (0,60)
60    FORMAT (29X, ‘INTRODUCTION b̸ TO b̸ COBOL’)
```

It is to be noted that apostrophes should be avoided within quotes since they will be misleading to the processor. (eg) An entry of literal type such as ‘RAMA’S ADVENTURES’ in a FORMAT statement will cause an error message while compiling.

Codes like I, F, E and D can be used in the same FORMAT statement as the literal data. (eg) The statements

```
      WRITE (0,55) AVRGE
55    FORMAT (10X, ‘AVERAGE=’, F6.2)
```

will result in displaying the literal ‘AVERAGE =’ followed by the value of the average, say as b̸ b̸ b̸ b̸ b̸ b̸ b̸ b̸ b̸ b̸ AVERAGE = 248.06.

10.4.2 Carriage Control Characters

In case of PRINT-OUT and screen, the first character in every line is removed from the total field and is used as a carriage control character. So it is essential to provide a carriage control character as the first character in the FORMAT statement to avoid the loss of first digit in the first variable or data to be printed. The carriage control characters and their functions are listed below

CARRIAGE CONTROL CHARACTER	CONTROL ACTION (FUNCTION)
“ ” (blank)	Advance by ONE line
“0” (zero)	Advance by 2 lines
“-”	Advance by 3 lines
“+”	Do not advance to a new line
“1”	Advance to top of next page.

The following examples will illustrate the usage of carriage control instructions.

EXAMPLE 10.12 Write the set of instructions for the processor to advance to top of next page and print a line containing 67 blanks followed by a literal ‘PAGE’ followed by a blank and an integer number of 3 digits

```
      WRITE (6,10) NPAGE
10    FORMAT (‘1’, 67X, ‘PAGE’, IX, I3)
```

The single blank after the literal may also be included in it and the above FORMAT statement can be written

```
10      FORMAT ('1', 67X, 'PAGE b̸', I3)
```

EXAMPLE 10.13 After printing the line given in Example 10.12, the processor has to advance by 2 lines and print 'DEPARTMENT OF COMPUTER SCIENCE—GOVT. OF INDIA', centred in that line. Assuming an 80 characters printer, write down the requisite instruction—

Solution:

```
        WRITE (6,20)
20      FORMAT ('0', 17X, 'DEPARTMENT b̸ OF b̸ COMPUTER
        b̸ SCIENCE b̸— b̸ GOVT. b̸ OF b̸ INDIA')
```

EXAMPLE 10.14 After printing the line in the Example 10.13, the processor has to advance by 3 lines and print the term 'EASTERN b̸ REGION b̸— b̸ CALCUTTA' centred in the line. Write down the requisite instruction.

Solution:

```
        WRITE (6,30)
30      FORMAT ('–' 27X, 'EASTERN b̸ REGION b̸ –b̸ CALCUTTA')
```

EXAMPLE 10.15 After printing the line in the Example 10.14, the processor has to advance to next line and print 'RESEARCH b̸ WING' after leaving 5 blanks as horizontal spacing. Write down the instruction.

Solution:

```
        WRITE (6,40)
40      FORMAT ('b̸', 5X, 'RESEARCH b̸ WING')
```

The above FORMAT can be written, including the first blank character, into the horizontal spacing and specifying it as 6x. The processor will compose a line with 6 blanks in the beginning, interpret the first blank as carriage-control instruction and, as such, advance by a single line and leave 5 horizontal blanks before printing the data. The rewritten FORMAT statement will be—

```
40      FORMAT (6X, 'RESEARCH b̸ WING')
```

EXAMPLE 10.16 After printing the line in the above problem, the processor has to leave 40 blank spaces and print 'SOFTWARE b̸ DIVISION' in the same line : Write down the instruction.

Solution:

```
        WRITE (6,50)
50      FORMAT ('+', 40x, 'SOFTWARE b̸ DIVISION')
```

(Exercise 12.4 may also be referred in this case.)

10.4.3 Use of "/" (Slash) in an Output Format Statement

The use of the "/" in an output FORMAT will indicate the end of one record or line image specification and the beginning of the next. Thus the same FORMAT statement can specify consecutive lines with different specifications. Also, the number of slashes to be used in-between will be one more than the blank lines to be provided and will be equal to line spacing. Example 10.17 will illustrate this.

EXAMPLE 10.17 Write down a single WRITE and FORMAT statement to print all the lines given in Examples 10.12 to 10.16, using slashes in the FORMAT statement.

Solution:

```
     WRITE (6,10) NPAGE
10   FORMAT (‘1’, 67X, ‘PAGE ƀ’, I3//17X,
     ‘DEPARTMENT ƀ OF ƀ COMPUTER ƀ SCIENCE’///
     27X, ‘EASTERN ƀ REGION ƀ-ƀ CALCUTTA’/
     6X, ‘RESEARCH ƀ WING’, 40X, ‘SOFTWARE ƀ DIVISION’
```

A slash in the middle of a FORMAT statement can be used to print alternate lines of same format—(eg)

```
     WRITE (6,10) I,X,J,Y
10   FORMAT (11X, I3, 5X, F7.2/10X, I3, 5X, F7.2)
```

will cause printing of values of variables I and X in lines 1, 3, 5 etc. and the values of J and Y in the other alternate lines say, 2, 4, 6 etc. The same way, slashes can be used to print every third line of same format.

Another use of slash is to print a single line of a given FORMAT and all the subsequent lines in another FORMAT. In such cases, the FORMAT of the subsequent lines should be enclosed by a set of INTERIOR PARENTHESES

```
     WRITE (6,10) N, A, B
10   FORMAT (38X, I4/(15X, F7.2, 5X, F6.2))
```

This will result in printing the value of N in the first line only and the values of A and B in the second and subsequent lines.

EXAMPLE 10.18 Write down the WRITE AND FORMAT statements to print the file PRIME NUMBERS in the first line and a list of prime numbers one in each in the subsequent lines—

```
     WRITE (6,10) NPRIME
10   FORMAT (6X, ‘PRIME NUMBERS’/(10X, I3))
```

Slashes can appear at the beginning or at the end of FORMAT statements (eg)

```
     WRITE (6,25) I, A
25   FORMAT (//// 6X, I3, 5X, F7.2//)
```

The above statements will result in 4 blank lines to precede the line in which values of I and A will be printed and also 2 blank lines after it.

10.4.4 T-Code

T-Code (TAB) can be used in place of X-Code. The T specification enables the programmer to position an output field anywhere in the print-line. (eg) The FORMAT statement in Example 10.12 can be written with T-Code as follows—

```
10   FORMAT (‘1’, T68, ‘PAGE ƀ’, I3)
```

Note that T 68 is equivalent to 67X. T 68 specifies that the literal ‘PAGE ƀ’ is to be printed beginning from column 68 onwards.

Just as in X-Code, if the T-Code is the first specification in the FORMAT, the first blank space will be taken as carriage control character. In such cases, the integer that follows the character T should be 1 excess of the TAB specification—(eg)

```
10   FORMAT (T36, ‘COMPUTATION’)
```

will cause the word COMPUTATION to be printed from columns 35 to 45.

It should be noted that the tabbing instructions of T-Code is always from the left margin as will be illustrated in the following example:

EXAMPLE 10.19 The value of I in the Format I4 is to be printed from column 10 onwards, value of A in the format F9.3 from column 25 onwards and the value of Y in the FORMAT E8.2 from column 40 onwards. Write the WRITE statement and the related FORMAT statement using the T-Code.

Solution:

```
        ||  WRITE (6,100) I, A, Y
100     ||  FORMAT (T12, I4, T25, F9.3, T40, E8.2).
```

EXAMPLE 10.20 Write down the FORMAT statement in the above example using X-Code in place of T-Code.

Solution:

```
100  ||  FORMAT (11X, I4, 10X, F9.3, 6X, E8.2)
```

The advantage in the T-Code is that the data fields need not be specified on the sequence they occur from left to right. The WRITE and FORMAT statements in Example 10.19 may also be written as follows—

```
        ||  WRITE (6,110) A, Y, I
110     ||  FORMAT (T26, F9.3, T40, E8.2, T11, I4)
```

10.4.5 Printing/Displaying of Numbers of I and F-Code Specification

The numeric data will be printed or displayed right justified in the specified columns. The leading zeros, if any, will be suppressed, ie, will be substituted by blanks.

As stated earlier I, F and E-codes are used in the FORMAT statements to describe the numeric fields. In case of I and F-Codes, if there is any possibility of a negative quantity, an extra column should be provided for printing it. If the field width specified is less, the most significant digits will be truncated. So, it is always advisable to specify a larger field-width. These will be illustrated in the case of I-Code in the following Table 10.1.

TABLE 10.1

DATA IN STORAGE	I-Code	DATA PRINTED
6246	I4	5246
−5246	I4	5246 (WRONG)
−5246	I5	−5246
5246	I5	b̸ 5246
5246	I3	246 (WRONG)
246	I5	b̸b̸ 246
−246	I6	b̸b̸ − 246

While using F-Code also, an extra column is to be specified for recording negative signs, if any. In this case also, it is safer to specify a larger field-width. Regarding the decimal positions, only the positions to the required accuracy are to be specified. (eg) If an amount in Rs and Ps is to be printed, it is enough if the decimal positions are given as 2. Any extra specification will result in trailing O's which will be unnecessary. (refer Table 10.2)

TABLE 10.2

DATA IN STORAGE	F-CODE	DATA PRINTED
458.868	F7.3	458.868
458.868	F8.3	ƀ 458.868
458.868	F6.3	58.868 (WRONG)
1468.85	F8.3	1468.850
468.85	F8.4	468.8500 *
−625.83	F7.3	−625.83
−625.83	F6.3	625.83 (WRONG)
−625.83	F5.3	−25.83 (WRONG)

* Please note that zeros will be supplied to fill out the vacant decimal positions of the numbers being printed.

10.4.6 Printing Numbers of E-code Specification

E-code is to be specified to print or display large or very small numbers, which are in REAL mode.

When an E-code is specified in the FORMAT related to a WRITE statement, the processor NORMALISES the number in the storage and displays it. The process of NORMALISATION shifts the decimal point to the left of the first significant digit and a leading zero precedes the decimal point which is preceded by a blank if the number is positive, or a negative sign, if negative. The normalised number is followed by the character E followed by 3 columns, one for recording the sign of exponent if it is negative. This will be blank if the exponent is positive. The other 2 columns are for recording the exponent whose value will determined by the number of positions by which the decimal point has been shifted next. In this case, not only a larger field width specification but also a larger number of decimal digit specifications will be advisable (refer Table 10.3).

TABLE 10.3

DATA IN STORAGE	E-CODE	DATA PRINTED
321.64	E12.5	ƀ 0.32164 Eƀ 03
−321.64	E12.5	−0.32164 Eƀ 03
321.64	E10.2	ƀƀ 0.32 Eƀ 03

If may be inferred from the above Table that, in the output field with E-specification, 7 positions have fixed roles. They are (1) the leading blank (2) the leading zero (3) the decimal point (4) the character E (5) the blank following E and (6 and 7) 2 positions for the exponent. So, the overall field with specification will be the number of significant digits to be printed + 7. (eg) If the data in the Exponential form is to be printed to the accuracy of 5 significant digits, the field width should be 12 (refer Table 10.3).

10.4.7 Self-Indexed Write Statement

Just as in the READ statement, self-indexing technique can be used in case of WRITE statement also. The statement should contain a starting (lower) index, a final (upper) index and index incrementation if it is other than 1. The final index should always be greater than the initial one (ie), index decrementation is not allowed. FORMAT statement, in this case also, gives the layout of a single line or record—(eg)

```
      WRITE (6,10) (N(I), I = 1,100)
10    FORMAT (4X, 12 (2X, I4))
```

This will result in printing of 100 numbers with 12 numbers per line. There will be 9 lines printed on all, with 8 lines each with 12 numbers and the 9th line with 4 numbers only.

Self indexing may be used to print 2 or 3 or more numbers alternately—(eg,)

WRITE (6, 10) (A(I), B(I), I = 1, 80) will result in printing of 80 values each of A and B alternately, while writing such arrays, DIMENSION statements have to be used. (refer 9.4.3.2)

10.5 END OF FILE CHECKING INCORPORATED READ STATEMENT

When a file contains a very large number of records, an END-OF-FILE indication is provided. It may be dummy END-CARD in case of card files and a TAPE-MARK in the case of Tape and disk files. This indication will be realised by the processor while reading the END-CARD or the TAPE-MARK. Hence, end-of-file (EOF) checking is incorporated in the READ instruction to be used in such cases. (eg)

```
        READ (5,10, END = 100) A, B, C, D
10      FORMAT (4 F7.2)
```

The term END = 100 means if it is end-of-file go to statement numbered as 100. In case of processing of files, there may be some EOF routines such as printing of totals.

EXAMPLE 10.21 A card file contains records of following format—

Column 1–5	Employee number (I5)
6–9	Employee identification (A4)
10–16	Pay (F7.2)
17–22	Bank A/C NO (I6)
23–80	Blanks

Write a program to copy the records on a floppy disk with the fields in the order: Bank A/C no., identification, Salary and Employee number.

Device number for card reader is 5, and for floppy 15. Provide for EOF checking in the READ instruction.

```
C       Program to copy Data From Card to Floppy
5       READ (5,10, END=50) NEMP, IDN, PAY, NAC
10      FORMAT (I5, A4, F7.2, I6)
        WRITE (15,20) NAC, IDN, PAY, NEMP
20      FORMAT (I6, A4, F7.2, I5)
        GO TO 5
50      STOP
        END
```

In the above program, when EoF condition is met, the processor is to go to statement 50 which instructs it to STOP.

Any FORTRAN program should end with an END statement preceded by a STOP statement. The significance of these statements are discussed later. (Chapter 13, Art. 2)

Summary

The sequence of operations in a computer is—(1) INPUT the Data, (2) Process Data and Compute the Result, and (3) Output the Result.

For inputting data, a READ statement is used. For outputing data, a WRITE statement is used. A READ or WRITE statement is accompanied by a FORMAT statement which gives the LAYOUT of data fields in the input record or gives how the output fields are to be configured on the screen or print-out or any other output medium.

In case of print-out, the first character that is met will be taken as a carriage control instruction.

Both READ and WRITE instructions are provided with self-indexing features so that a number of similar fields can be read or written with a single READ or WRITE instruction. To facilitate processing files with large number of records, the END-OF-FILE checking facility is incorporated in the READ instruction.

Exercise/Questions

10.1 Write a READ statement to read from a record the values of variables N, A, B, C. The input device reference number is 5.

Write down the associated FORMAT statement if in a typical record

N = 4863

A = RAMA

B = −4.82

C = 0.473E36.

10.2 Write a single statement to read 100 numbers, placing the first 50 numbers in the Array A and the other 50 in the Array B. Assume the device number.

10.3 Given 100 records each containing 5 variables, M, A, B, C, D, write a self-indexed READ statement to read them. The input device reference number is 4.

10.4 There are 16 cards each containing 16 integer numbers, each of 5 digits. Write a READ statement to read odd subscripted numbers from 17th position to 199th position. Write down the FORMAT statement also.

10.5 Write FORMAT statements for reading REAL numbers of following descriptions. Assign statement number to the FORMAT statement.

1. 8 digit number with no decimal place
2. 10 digit number with 4 decimal places
3. 7 digit number with 6 decimal places.

10.6 Write the READ and FORMAT statements to read a Tape-record of format

Character 1–5	NMATL
6–10	NREF
11–17	WHOLESALE RATE IN Rs and Ps
18–24	RETAIL RATE IN Rs and Ps

10.7 500 numbers each of FORMAT E10.4 are recorded 8 numbers per card. Write down the READ and FORMAT statements. How many members will the last record contain?

10.8 200 sets of values of A(I), B(I) and C(I) are to be printed with one set per line. Write a WRITE statement and a FORMAT statement. Assume a suitable F-Code for A, B and C.

10.9 Write an OUTPUT statement to centre the literal 'BANGALORE UNIVERSITY' and display it in an 80 column screen.

10.10 Draw the flow-chart for the program under Example 10.1.

for NOTES

Chapter Eleven

Arithmetic Statements and Library Functions

11.1 PROCESSING STATEMENTS

The processing statements are of 2 major types, namely—(i) ARITHMETIC STATEMENTS, and (ii) CONTROL STATEMENTS. Arithmetic statements are discussed in this chapter.

11.1.1 Arithmetic Statement

The Arithmetic statement is used to instruct the computer to perform a mathematical computation. These computations can be made by the computer both in the INTEGER (Fixed Point) and REAL (Floating Point) modes. It is preferable to use the REAL mode for Mathematical operations in view of lack of accuracy in the INTEGER mode. (eg) 5/2 in the INTEGER MODE will be 2 whereas in the REAL MODE 2.5.

11.1.2 Arithmetic Operations and Operators

FORTRAN provides for all the basic Arithmetic operations. The operations and their operators are listed in the Table 11.1

TABLE 11.1

OPERATION	OPERATOR
ADDITION	+
SUBTRACTION	–
MULTIPLICATION	*
DIVISION	/
EXPONENTIATION	**

Besides the above operators an Arithmetic statement contains an ASSIGNMENT SYMBOL (=) and may often use parentheses, ie, Semicircular Brackets.

11.1.3 Arithmetic Expression

An Arithmetic Statement consists of a variable name called IDENTIFIER, followed by an Assignment Symbol (=), followed by an Arithmetic Expression.

In FORTRAN, $A = B$ means Assign the value of B to A. The value of A will be changed to that of B. But the value of B will remain unchanged. (eg) If value of A has been 8.2 and that of B 6.48 before the execution of the statement $A = B$, after its execution, A will become equal to 6.48, but B will remain of same value (ie) 6.48.

The Arithmetic Expression may be a single constant or a variable or a combination of variables and/or constants and Arithmetic operators. (eg,)

$I = 6; J = -5, A = 6.2, B = -5.84, A = B + C * D - F/G + 6.1, M = I - K * N - 8.$

The Arithmetic expression should contain quantities (variables and constants) on the same mode (ie) all in the INTEGER mode or in the REAL mode. Mixed mode of operations are not permissible in FORTRAN, except for exponents.

The processor computes the value of the Arithmetic expressions and assigns the computed value to the identifier.

The identifier may be in the same mode as the quantities in the Arithmetic expression or in a different mode.

The Arithmetic expression is computed in its mode and assigned to the IDENTIFIER [variable to the left of equal (assignment) sign] converting its mode if indicated.

For example,

If $A = 4.28, B = 3.64, C = A + B$ will be executed as follows—4.28 will be added with 3.64 and the result will be, as it is, assigned to C (ie) $C = 7.92$.

Similarly, if $N = 8, M = 6, K = N - M$ will be executed as follows. 6 will be subtracted from 8 and the result will be, as it is, assigned to K (ie) $K = 2$.

If $I = A + B$ with the above values of A and B, the sum of A and B will be computed as 7.92 (in the REAL mode), converted as 7 in the INTEGER mode and assigned to I, (ie) $I = 7$.

Similarly, $D = N - M$ will be executed as follows. The result of $8 - 6$ namely 2 will be converted with REAL mode as 2.0 and assigned to D, (ie) $D = 2$.

11.1.4 Hierarchy of Operations

Hierarchy of operations means the order in which the Arithmetic operations will be carried out by the computer.

Since the computer can execute only one operation at a time, it is designed to execute the operations in an hierarchical order.

The FORTRAN rules of hierarchy consists of three parts which are called 'levels'.

Level 1. All exponentiation (if any) is executed first.

Level 2. All multiplications and/or divisions (if any) is executed next.

Level 3. All additions and/or subtractions (if any) is done last.

All operations on a given level will be completed before going to next level. The operations on the same level will be executed in the sequence they occur from left to right.

In the case of sequential exponentiation operators (eg) x * *y * *z, the execution will be from right to left. (ie,) y * *z will be evaluated first and then x will be raised to the power (y * *z). But if there is an expression A * *B + C * *D, A * *B will be evaluated first and then C * *D.

EXAMPLE 11.1 Write down the FORTRAN form of the equation $S = ut + \frac{1}{2}at^2$.

Solution:

$$S = U * T + A * T * *2/2.0.$$

Note that UT does not mean U multiplied by T in computer language; it should be written as U*T.

The exponent 2 in T^2 can be written in INTEGER form. It is not necessary to write it as 2.0 (ie) in the REAL form.

The denominator 2 anyhow has to be written in the REAL form (ie) as 2· or as 2.0 to avoid mixing of modes.

Though the above expression is correct, it can be rewritten as:

$$S = U * T + A * T * T/2.0,$$

since execution of T*T is faster than T**2.

EXAMPLE 11.2 Trace the sequence of execution of the statement D = C + B * x + A * x * *2.

Solution:

(i) x * *2 (ie) x^2 is evaluated first

(ii) B * x (ie) Bx is evaluated next

(iii) A * (x^2) (ie) Ax^2 is evaluated next

(iv) C + Bx is evaluated next

(v) (C + Bx) + Ax^2 is evaluated next

(vi) The result evaluated is assigned to D.

11.1.5 Use of Parenthesis

Parentheses have to be used in Arithmetic expressions when the normal hierarchy of operations has to be altered to get the desired result. (eg) The expression A/(B + C) has to be written as A/(B + C) since A/B + C will be evaluated as $\frac{A}{B} + C$. The parentheses enclosing A + B in this case informs the processor to evaluate (A + B) first instead of A/B.

When an Arithmetic expression contains a pair of parentheses the sub-expression within the parentheses will be evaluated first and then the operations outside the parentheses (eg) in the expression A * (B + C)/D the sequence of evaluation will be —

1. Addition of B and C
2. Multiplication of A with (B + C)
3. Division of A * (B + C) by D.

It should be noted that the left parentheses does not indicate multiplication in COMPUTER-LANGUAGE: (eg)

A(B + C) does not mean A multiplied by (B + C) in COMPUTER language. This should be written as A * (B + C).

The sub-expressions within parentheses may contain 2 or more operations. (eg)

$$A + B/(C * D + E + F * *2) - G$$

The sub-expressions within any pair of parentheses will be evaluated according to the hierarchy of operation. In the above example, the sub-expression within the parentheses $(C * D + E + F * *2)$ will be evaluated by the following sequence

1. Exponention (F^2)
2. Multiplication ($C * D$)
3. Addition 1 ($C * D + E$)
4. Addition 2 $[(C * D + E) + F^2]$

An expression may contain more than one sub-expression each enclosed within a pair of parentheses. In such cases, the evaluation of parenthesised sub-expressions is done in the sequence they occur from left to right. (eg) $A * (B/D) + E/(F * G) - H$

In this case the parenthesised sub-expression B/D is evaluated first and then the other parenthesised sub-expression $F * G$.

11.1.5.1 Nested Parentheses

A pair of parentheses completely contained within another pair is called a NEST of parentheses. (eg) (A+B*(C+D)+E)

A nest of parentheses may contain any number of such pairs each occurring within another pair. (eg,)

$(A + (B * C)/(F - G) - H)$ has a nest of 3 parentheses.

When sets of parentheses are nested, the operations within the innermost parentheses are evaluated and using this result the expression enclosed in the immediately next pair of outer parentheses is evaluated and so on. In the above example:

(i) $(F - G)$ is evaluated first

(ii) $(B * C/(F - G))$ is evaluated next

(iii) The expression within the outermost parentheses is evaluated last.

The general procedure in case of nested parentheses are:

(a) The execution starts with the evaluation of sub-expression within the innermost of pairs of parentheses and proceeds to evaluations of terms within the immediately next pair of outer parentheses and ends with evaluation of terms within the outermost parentheses.

(b) Parenthesised expressions are evaluated first before other operations are executed.

(c) The normal rules of hierarchy are applicable to the sub-expressions within any pair of parentheses.

(d) The result of the sub-expressions within parentheses will become the quantity to be used in the over-all expressions.

11.1.6 General Guide-Lines to Write Arithmetic Statements

(1) Avoid writing a too long statement which may result in wrong form and wrong computation.

The Arithmetic expressions is to be split into a number of instructions each involving only one operation by the compiler. Only these instructions are to be converted into machine code and stored in the memory. As such, there is no sanctity in writing too lengthy Arithmetic statements. Any lengthy statement may be divided into 2 or more statements. This will reduce errors in statements and, hence, errors in results. Also checking of statements will be easier.

(2) Avoid as far as possible mixed mode statements.

(3) Use parentheses not only wherever necessary but also whenever in doubt. Excess parentheses will do no harm but absence of parentheses may lead to wrong result. Also the parentheses will be dropped during compiling and will not find a place in the object program.

(4) Avoid redundant calculations. (ie,) If a sub-expression is to be used a number of times, evaluate if once and use it wherever necessary. (eg) In the Example 11.3, the expression $A_1*B_2-A_2*B_1$ is to be used twice, once for evaluating the value of x and next for evaluating y. So it is evaluated as $D = A_1 * B_2 - A_2 * B_1$ and the value of D is used in calculation of x and y. This saves computer time.

(5) Prefer addition to multiplication wherever possible since execution of addition is faster than multiplication. Execution of (A + A) is faster than (2.0 * A).

(6) Prefer multiplication to exponentiation wherever possible since execution of multiplication is faster than exponentiation. (T * T) will require less execution time than (T * *2)

(7) Unless it is necessary, see that the exponent is an INTEGER. T * *4 will be executed faster than T * *4.0.

T * *4 will be computed as: T*T*T*T

T * *4.0 will be computed as:

ANTILOG (4.0 * LOG(T))

EXAMPLE 11.3 A pair of simultaneous equations in x and y are of the form $A_1x + B_1y = C_1, A_2x + B_2y = C_2$.

The values of $A_1, B_1, C_1, A_2, B_2, C_2$ are given in a line through the keyboard each of format F7.2 (inclusive of negative sign if any)

Write a program to display the values of x and y on the screen.

Solution:

$$X = (C1B2 - C2B1)/A1B2 - A2B1), Y = (A1C2 - A2C1)/(A1B2 - A2B1).$$

Program

```
C     SOLUTION TO SIMULTANEOUS EQUATIONS IN X AND Y
      READ (0,5) A1,B1,C1,A2,B2,C2
    5 FORMAT (6F7.2)
      D = A1 * B2 - A2 * B1
      X = (C1 * B2 - C2 * B1)/D
      Y = (A1 * C2 - A2 * C1)/D
      WRITE (0,10) X,Y
   10 FORMAT (6X, 'X=', F7.2, 5X, 'Y=', F7.2)
      STOP
      END
```

11.2 LIBRARY FUNCTIONS

ARITHMETIC STATEMENTS can be used to perform calculations involving Arithmetic operations only. But solution to scientific and technical problems will often involve calculation of trigonometric and or logarithmic functions. Such functions can be evaluated by the use of Arithmetic operations with the help of certain numerical methods, (eg) sin(x) may be evaluated using the series—

$\sin(x) = x - (x^3)/(3!) + (x^5)/(5!) - (x^7)/(7!)$ etc.

(where x is in radians.)

Writing such program steps whenever a trigonometric or logarithmic function is to be used will result in enormous duplication. As such, most of the programming languages provide package of previously written programs to compute such functions whenever necessary in the user's programs. The package of such programs is called 'LIBRARY' and these programs contained in the library are called Built-in-Functions or LIBRARY FUNCTIONS. A library function is made up of a function name followed by one or more ARGUMENTS within parentheses. (eg) EXP (A); in this, EXP is the function name and A is the argument. The Library Functions are also called as OPEN SUB-PROGRAMS.

11.2.1 Trigonometric and Logarithmic Functions

Table 11.2 gives the list of trigonometric and logarithmic functions available in FORTRAN IV.

TABLE 11.2

FUNCTION	MODE OF FUNCTION	MODE OF ARGUMENT	COMMENTS
sin (A)	REAL	REAL	A in radians
cos (B)	REAL	REAL	B in radians
tan H(D)	REAL	REAL	D in radians
A tan(E)	REAL	REAL	$\tan^{-1}$ E (Result will be in radians)
EXP (G)	REAL	REAL	e^G
ALOG (H)	REAL	REAL	NATURAL LOG
ALOG 10 (P)	REAL	REAL	LOG TO BASE 10 (Common LOG)

It should be noted that the Function names are according to the 'naming Rules' and hence the 2 log functions are referred as ALOG and ALOG10 since they are in REAL mode. There is no library function to compute ANTILOG (refer Examples 11.6 and 11.9).

For trigonometric functions, only natural SIN and COS are available in most of the implementations. Other functions, TAN, COT, SEC and COSEC, have to be derived using these two. Similarly, for inverse functions ARC TAN, $(\tan^{-1})$ alone is available and for Hyperbolic only TAN H is available.

EXAMPLE 11.4 The value of an angle in degrees is in the format F7.3 (eg, 36.483°) is input through the key-board. Write a program to compute COSINE of the angle.

Program

```
C     GIVEN AN ANGLE IN DEGREES TO FIND ITS COSINE
      READ (0,5) A
    5 FORMAT (F7.3)
      B = COS (A/57.29578)
      WRITE (0,10) A, B
   10 FORMAT (6X, 'COSINE OF ANGLE b̸', F7.3, 1X,
      'DEGREES IS b̸' F7.5)
      STOP
      END
```

(1 RADIAN = 57.29578 degrees)

EXAMPLE 11.5 A card contains the following readings recorded during an experiment

COLUMNS 1–5	VOLTAGE (F5.1)
6–10	CURRENT (F5.2)
11–17	POWER (F7.2)
18–80	BLANKS

Write a program to compute the power factor (pf) and pf angle in degrees and print them.

Solution:

$\text{pf} = P/V * C = \cos\varphi$

(P = POWER, V = VOLTAGE, C = CURRENT)

$\text{PHI} = \Phi = \text{pf angle} = \cos^{-1}(\text{pf})$ in radians

Program

```
C     TO COMPUTE PF ANGLE IN AN A.C. CIRCUIT
      READ (5,10) V, C, P
   10 FORMAT (F5.1, F5.2, F7.2)
      PF = P/(V * C)
      T = SQRT (1-PF * PF)/PF
C     T = TAN (PHI)
      PHI = ATAN(T) * 57.29578
      WRITE (6,15) PF, PHI
   15 FORMAT (6X, 'THE POWER FACTOR=,' F7.5,
      5X, 'PF ANGLE=,' F7.3, '/p DEGREES,'
      STOP
      END
```

EXAMPLE 11.6 The value of A, which is natural log of B, is given through the keyboard in the format F7.5. Write a program to print the value of B (Antilog of A).

Program

```
C     TO COMPUTE ANTILOG, GIVEN NATURAL LOG
      READ (0,5)A
    5 FORMAT (F7.5)
      B=EXP(A)
      WRITE (6,10),A,B
   10 FORMAT (6X,'THE ANTILOG OF b̸', F7.5,
      'b̸ IS b̸', F7.2)
      STOP
      END
```

EXAMPLE 11.7 The value of x which is common log of y is given through the key-board in the format F7.4. Write a program to print the value of y.

```
C     GIVEN COMMON LOG TO FIND ANTILOG
      READ (0,10) x
   10 FORMAT (F7.4)
      y=10.0**x
      WRITE (6,15), x,y
   15 FORMAT (6x, 'THE ANTILOG OF /p', F7.4.
      '/p IS /p, F9.2)
      STOP
      END
```

11.2.2 Complex Functions

FORTRAN IV has provisions for programming with complex numbers. Only real variables should be used as complex numbers. The complex variables should be declared by a type statements. (eg)

```
      COMPLEX A,B,C
```

Any complex variable is recorded always with its REAL part first followed by the imaginary part. Negative sign, if any, has to be recorded. The operator i or j is not recorded at all (refer Table 11.3)

TABLE 11.3

COMPLEX NUMBER	ENTRY FORM IN INPUT RECORD TO COMPUTER	
4.5 + i 5.2	4.5	5.2
– 4.5 + i 6.8	– 4.5	6.8
5.2 – i 4.8	5.2	– 4.8
– 5.6 – i 7.2	– 5.6	– 7.2

When a number is declared as COMPLEX, the 2 adjacent code definitions in the Format statement will be taken as the definition for the real and imaginary parts, (eg,)

```
      COMPLEX A
      READ (0,5) A
    5 FORMAT (2F7.2)
```

COMPLEX constants are also to be declared by type statement. They are entered with the numerical values of real part followed by that of imaginary part. These are separated by a comma and enclosed within parentheses,

(eg,)

```
COMPLEX Z
Z = (4.8, 3.6)
```

The arithmetic operations on complex numbers are all represented by the same operators as for the REAL or INTEGER numbers.

The basic library functions used in operation of complex variables are (1) CABS(x), (2) REAL(x), (3) AIMAG(x) where x is a complex number (refer 14.2.4.1)

CABS(x) gives the absolute value of x, REAL (x) gives its REAL part, and AIMAG(x) gives its imaginary part.

If $x = (4.8, 3.6) = 4.8 + i3.6)$ and if $y =$ CABS $(x) = \sqrt{4.8^2 + 3.6^2} = 6.0$

IF $Z =$ REAL$(x) = 4.8$

IF $W =$ AIMAG $(x) = 3.6$

(refer Example 14.7 under Section 14.2.4.1)

11.2.3 Other Library Functions

Besides the library functions discussed above, there are some more library functions such as SQRT which go a long way in the reduction of instructions in the user's programs. Table 11.4 gives a list of such functions.

TABLE 11.4

FUNCTION	MODE OF FUNCTION	MODE OF ARGUMENT	NUMBER OF ARGUMENTS
SQRT(A)	REAL	REAL	1
ABSOLUTE VALUE FUNCTIONS:			
ABS (B)	REAL	REAL	1
IABS (J)	INTEGER	INTEGER	1
MODE CONVERSION FUNCTIONS:			
FLOAT (K)	REAL	INTEGER	1
IFIX (D)	INTEGER	REAL	1
TRUNCATION FUNCTIONS:			
AINT (E)	REAL	REAL	1
INT (E)	INTEGER	REAL	1
MODULO REMAINDERING FUNCTIONS:			
MOD (M,N)	INTEGER	INTEGER	2

For using the SQRT function, the argument should not only be a REAL variable but should not be negative also, (eg,)

$$H = SQRT(A * A + B * B)$$

The ABSOLUTE value functions give the magnitudes of the arguments, regardless of the original sign of the argument quantity.

The function ABS is used if the argument is REAL and IABS if it is INTEGER. (eg,)

If A = ABS(B)

and $B = 7.82$; A will be $= 7.8$. If B was -7.82, A will be $= 7.82$.

Similarly, if I = IABS(J),

and $J = 8$, I will be 8; If J was -8, I will be 8.

The Float function is used to convert a number in the INTEGER mode to REAL mode. (eg,)

A = FLOAT(K)

If K = 8 A will be 8.0

If K = −7 A will be −7.0

Conversely, the IFIX function is used to convert a number in the REAL mode to INTEGER mode. During this conversion the decimal portion of the real number is truncated. (eg,)

K = IFIX(D)

IF D = 8.62, K will be = 8

IF D = 8, K will be = 8

IF D = −8.52, K will be = −8

It should be noted that conversion of mode is permitted only on variables defined implicitly and not on variables with explicit definition.

The truncation functions are used to remove the fractional portions of numbers. (eg,)

If F = AINT(E)

If E = 8.625, F will be = 8.

If E = −6.42, F will be =−6.

The INT function is IFIX by another name. It is provided just as a companion to AINT.

The MOD function has 2 arguments both of which are integers.

If K = MOD(M, N) it means K is the remainder left after dividing M by N.

If M = 23 and N = 4, K will be 3.

This function is useful to check whether a number is odd or even.

Any number of library functions can be used in an Arithmetic expression/statement.(eg)

J = IFIX(SQRT(FLOAT(N)))

A = ALOG10(EXP(SIN(x)))

Also an Arithmetic expression/statement can contain one or more library expression in addition to Arithmetic operations. (eg)

$$A = B * C + EXP(F/D) * COS(y) - P/Q$$

11.3 USEFULNESS OF ARITHMETIC STATEMENTS

The Arithmetic statements are used

(i) to evaluate an expression and assign that value to an identifier. (eg,)

$X_1 = (-B + SQRTCD)/(A + A)$

(ii) To initialise variables. (eg,)

NCTR = 0

(iii) To change the values of variables. (eg,)

N = N + 5

M = M − 2 and

(iv) to provide steps to solve a mathematical formula. (eg,)

DR = A + A

DIS = B * B – 4.00A * C

X2 = (–B – SQRT(DIS))/DR

Summary

A FORTRAN Arithmetic statement is to instruct the computer to perform one or more Arithmetic operations to compute an entire expression. The operations will anyhow be performed one at a time. They will be carried out in the specific order: exponentiation, multiplication and/or division, and, addition and/or subtraction in each case moving from left to right in the Arithmetic expressions until all operations at a given level of hierarchy are completed. Most of the Arithmetic operations are to be done in the REAL mode in view of better accuracy. But exponents and counting operations may be in the INTEGER mode.

To evaluate trigonometric and logarithmic functions, library functions are available in FORTRAN. They are previously written programs using some numerical method or other, and will be available to the user's program at execution time. They are called into action wherever their particular names occur in the user's programs.

Besides trigonometric and logarithmic functions, there are built-in-functions for operations on complex numbers and other functions such as SQRT, MOD, FLOAT, IFIX etc.

Exercise/Questions

11.1 Identify each of the following statements as INTEGER or REAL or MIXED MODE

(i) A = 2.0 * B * C + (E – F) * 3
(ii) A = B * SIN(x) – EXP(y)
(iii) I = IFIX(A + B * D – C/E)
(iv) A = FLOAT(K * 4 – J * M/N)
(v) M = N/K – L * J
(vi) N = A * B – C/D
(vii) X = I/J * (M – N)

11.2 Write the following Arithmetic statements in FORTRAN FORM:

(i) $V = \frac{4}{3}\pi r^3; (\pi = 3.141592)$
(ii) $C = 2R \sin \frac{A}{2}$
(iii) $R = \frac{\sin^3 x \cos^2 x}{3} + \frac{2}{15} \sin^2 x$
(iv) $G = \frac{1}{2} \log \frac{1+\sin x}{1-\sin x}$
(v) $z = \frac{1}{\sqrt{x^2-A^2}} - \frac{2A^2}{3(\sqrt{x^2-A^2})^3}$

(vi) $A = 2e/r^2$

(vii) $b_1 = a_1 + a_2x_1 + a_3x_1^2$

11.3 IDENTIFY the errors in the following FORTRAN Statements

(i) $-V = A + B$

(ii) $V * V = U * U + 2.0 * G * S$

(iii) $F = M * A$

(iv) $A = A + 5$

(v) $I = I - 2.72$

(vi) $I + 2 = J - 3$

11.4 Evaluate the following expressions given that $B = 4.82, C = -3.62$, $D = 1.5, E = 2.65$ and $F = 3.2$ and $I = 3, M = 1, N = 2, L = 3$ and $K = 4$

(i) $A = B * C - D/E + F$

(ii) $J = I/M + N * L - K$

(iii) $A = M * I + L/N + K$

(iv) $I = B/C + D * E - F$

Write down the value and the mode of the result.

11.5 Write a program to read temperature in FAHRENHEIT and convert it into CELSIUS.

11.6 Write a program to read temperature in CELSIUS and convert it into FAHRENHEIT.

for NOTES

Chapter Twelve

Control Statements

12.1 CLASSIFICATION OF CONTROL STATEMENTS

The Control Statements are of 2 types—(i) PROGRAM CONTROL STATEMENTS, and (ii) PROCESSOR CONTROL STATEMENTS.

12.2 PROGRAM CONTROL STATEMENTS

The normal order of execution of statements by the computer is the order in which they are written in the program. Program Control Statements have to be used wherever this normal order has to be altered.

A (Program) Control Statement is defined as a statement used to make the computer execute a statement other than the next statement in sequence.

The (Program) Control Statement is of 2 types—(a) Branch Statements, and (b) Loop Statements.

This chapter discusses the Branch Statements which are also called 'transfer of control' statements, or Jump statements.

12.3 TRANSFER OF CONTROL OR BRANCH STATEMENTS

These statements can be classified as—(1) Unconditional Branch Statements, and (2) Conditional Branch Statements.

12.3.1 Unconditional Branch Statement

FORTRAN provides 2 types of Unconditional Branch Statements, namely (1) GOTO, and (2) ASSIGNED GOTO. These are also called 'UNCONDITIONAL TRANSFER' Statements.

12.3.2 GOTO Statement

The GOTO Statement provides a means of transferring control to some statement other than the next one in sequence, without imposing any condition.

It consists of two words 'Go To' followed by a single statement number (eg) GOTO 40. This tells the computer to go next to the statement specified by the number 40.

GOTO statement can be used to branch back to an earlier statement (refer Example 10.1) or to skip some portion of the program and branch ahead to a latter statement. (refer Example 12.2) As in example 10.1, GOTO statement is often used to return from the end of program to the beginning which is essential while processing a number of records using the same program. Note that GOTO statement should not be addressed to a FORMAT statement.

Assigned GOTO will be discussed later in this chapter (Section 12.6)

12.4 CONDITIONAL TRANSFER OF CONTROL STATEMENTS

These are also called CONDITIONAL branch or DECISION MAKING statements.

The Conditional Branch Statements in FORTRAN are: (i) ARITHMETIC IF (ii) LOGICAL IF, and (iii) COMPUTED GOTO.

12.4.1 Arithmetic IF Statement

An Arithmetic IF statement consists of the word "IF" followed by a pair of parentheses containing a variable or any desired expression, after which come three statement numbers.

(eg) IF (D) 10, 15, 20

IF (x − y)15, 10, 5

Such a statement means compute the expressions within parentheses. If the computed value is negative, GOTO the first indicated statement; else if the value is ZERO, GOTO the second indicated statement and if it is positive GOTO the third indicated statement.

(eg) Taking the statement IF (D), 10, 15, 20, it means, if D is negative GOTO Statement 10, or if D is Zero GOTO statement 15, or if D is positive GOTO statement 20.

The expression whose value is to be tested in the Arithmetic IF may be simple or complex. It may contain parentheses of its own; but it must be completely contained in a pair of parentheses belonging to the Arithmetic IF statement itself.

(eg) IF (A * *2) 5,10,15

IF (I * *3 − J)20, 25, 30

IF ((A + B)/C)40, 35, 50

IF ((A(I) * *2 − I) + S)65, 60, 55

In an Arithmetic IF statement, two of the number options may be the same

(eg) IF (I) 10, 15, 10

This means, if I=0 GOTO Statement 15; else GOTO Statement 10

IF (A) 20, 25, 25

This means, if A is negative GOTO 20; Else GOTO 25.

IF (x − y)30, 30, 35

This means, if (x − y) is positive GOTO statement 35. Else GOTO 30.

Note that in an Arithmetic IF Statement, if all the three number options are the same, it will make no sense.

EXAMPLE 12.1 Two integer numbers each of maximum 3 digits are to be given as input through key-board. Write a program to compute their GCD and LCM and display them on the screen.

Solution:

The algorithm to find the GCD of 2 integer numbers M and N has been discussed in Chapter 8 (refer Section 8.3.2)

$$LCM = M * N/GCD$$

It should be noted that the original values of M and N are to be preserved to compute them LCM.

Program

```
C       TO COMPUTE GCD AND LCM OF 2 INTEGER NUMBERS
C       USE OF ARITHMETIC IF
        INTEGER GCD
        READ (0,5) M,N
    5   FORMAT (2 I 3)
C       SAVE VALUES OF M AND N
        I=M
        J=N
C       COMPUTE GCD USING I AND J
   10   IF (I-J) 15, 25, 20
   15   J=J-I
        GOTO 10
C       THE ABOVE IS AN EXAMPLE OF GOTO
C       STATEMENT TO BRANCH TO A PREVIOUS STATEMENT
   20   I=I-J
        GOTO 10
   25   GCD=I
        LCM = M * N/GCD
        WRITE (6,30) M,N,GCD,LCM
   30  0FORMAT (6x, 'M=', I3, 5x, 'N=', I3,
       110x, 'GCD=', I3, 5x, 'LCM=', I 6)
        STOP
        END
```

The algorithm used in the above solution is a modified version of EUCLID's algorithm.

12.4.2 Logical IF Statement

The logical IF statement provides a choice between two alternatives depending on whether a specified condition is satisfied or not. The condition is specified in the form of 'LOGICAL EXPRESSION' (eg)

(A.GT.B).

These expressions use any one of the following "RELATIONAL OPERATORS". Note that these operators must always be preceded and followed by a period (full-stop):

OPERATOR	RELATIONAL OPERATION
.GT.	Greater than
.GE.	Greater or Equal
.LT.	Less than
.LE.	Less or equal
.EQ.	Equal
.NE.	Not Equal

The logical IF statement is of the form:

IF (LOGICAL EXPRESSION) Statement–1, Statement–2.

(Statement–2 will be the next statement in sequence),

Statement–1 will be of 2 types (a) Computing Statement, and (b) Control Statement.

(eg) IF (A.GT.B)D = A + B

C = D – 2.8

In this example, Statement–1 is a computing statement, namely $D = A + B$.

$C = D - 2.8$ is the next sequential statement.

The above IF statement means if $A > B$, compute $D = A + B$, and then compute $C = D - 2.8$.

Else (if $A \not> B$) straight compute $C = D - 2.8$ (refer FLOW-CHART in Fig 12.1.)

The Statement–1 may be a control (ie) GOTO statement as illustrated below:

IF (I.GE.J) GOTO 20

J = J – I

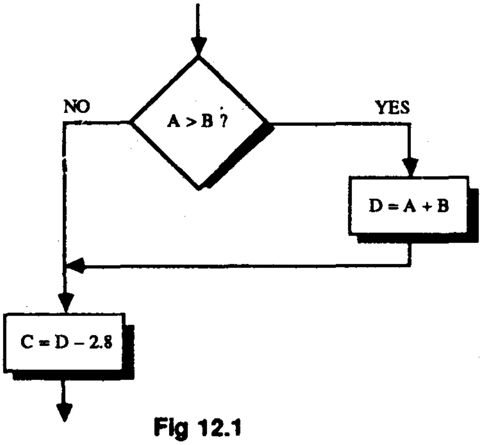


Fig 12.1

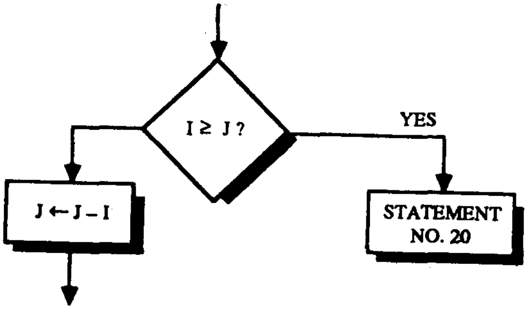


Fig 12.2

The above pair of statements means if $I \geq J$ GOTO statement 20; (else) make $J \leftarrow J - I$.

This is illustrated in flowchart in Fig 12.2.

The logical expression, as seen in the above examples, tests the relation between 2 variables. It may also test the relation between a variable and a constant. In any case, the 2 operands of the relational operators should be in the same mode. (eg)

IF (A.LT.B)
IF (I.LE.J) } are valid

$\left.\begin{array}{l}\text{IF (A.EQ.J)}\\ \text{IF (I.NE.B)}\end{array}\right\}$ are valid

$\left.\begin{array}{l}\text{Also IF (A.EQ.FLOAT(J))}\\ \text{IF (I.NE.IFIX(B))}\end{array}\right\}$ are valid

$\left.\begin{array}{l}\text{IF (A.EQ.10.0)}\\ \text{IF (I.LE.8)}\end{array}\right\}$ are valid

$\left.\begin{array}{l}\text{But IF (A.EQ.10)}\\ \text{IF (I.EQ.11.2)}\end{array}\right\}$ are valid

While using constants in logical expressions they should not be the first operands.

Statements like IF (10.0.EQ.B) and IF (8.NE.K) are invalid.

EXAMPLE 12.2 For plotting a graph of safe-load S on a column with reference to the slimness ratio R, the following empirical formulas are to be used:

$S = 17000 - 0.485R^2$ if $R < 120$; else $S = 18000/(1 + R^2/18000)$

It is desired to obtain values of S for values of R from 60 to 180 at intervals of 10. Write a program for the above job.

Program

```
C         SAFE-LOADS S ON A COLUMN
C         USE OF LOGICAL IF
          R=60.0
      5   IF(R.GE.120) GOTO 10
C         ELSE
          S=17000.0 - 0.485 * R * R
          GOTO 15
C         THE ABOVE IS A GOTO STATEMENT TO BRANCH 'AHEAD' TO A LATER
          STATEMENT
     10   S=18000.0/(10+R * R/18000.0)
     15   WRITE (6,20) R,S
     20   FORMAT (6x, F5.2, 5x, E18.8)
          R=R+10.0
          IF (R.LE.180) GOTO 5
C         ELSE
          STOP
          END
```

12.4.2.1 The LOGICAL IF has the following useful applications

(1) Decision making; (eg) IF Statement numbered as 5 in the program under Example 12.2.

(2) To branch back from the end of the program to the beginning, so as to repeat the execution of the program with different input or data (eg) IF statement before the STOP statement in the program under Example 12.2.

(3) To get out of a loop (refer Section 13.5 and Example 13.6)

(4) In iterative operations (refer Example 12.3)

In iterative operations, an initial value of the result is guessed. Using this a more accurate result is computed. One such process of computation is called ITERATION. After each iteration, the values of results before and after the iteration are compared. If both are the same, or if the absolute value of the difference between them is too small, the latest value of the result is given as output. Else, using the latest value as previous value, the iterations are continued. The Example 12.3 illustrates this.

EXAMPLE 12.3 Write a program to compute any one of the roots of the equation $x^3 - 3x^2 + 4x - 5 = 0$ by NEWTON's method.

Solution:

If x_1 is an approximate root of an equation $f(x) = 0$, a more accurate root x_2 is given as:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

$$f(x_1) = x_1^3 - 3x_1^2 + 4x_1 - 5$$

$$f'(x_1) = 3x_1^2 - 6x_1 + 4$$

Program

```
C         TO FIND ANY ONE ROOT OF A CUBIC EQUATION
C         USE OF LOGICAL IF IN ITERATIVE OPERATIONS
C         GUESS AN INTIAL VALUE OF X1
          X1 = 1.0
   5    0 FX1 = X1 * *3 - 3.0 * X1 * X1
        1 +4.0 * X1 - 5.0
          FDX1 = 3.0 * X1 * X1 - 6.0 * X1 - 4.0
          X2 = X1 - FX1/FDX1
          IF (ABS(X2 - X1). LT. 0.00001) GOTO 10
          ELSE
          X1 = X2
          GOTO 5
  10      WRITE (0,15) X2
  15      FORMAT (6X; 'ONE ROOT OF GIVEN EQUATION=', F10.4)
          STOP
          END
```

The above method is also known as NEWTON-RAPHSON method.

In case of REAL variables, LOGICAL IF is to be preferred to ARITHMETIC IF. This is because, to check for ZERO in case of REAL numbers, the approximation may result in a non-zero value when a zero value is actually expected and hence a wrong branching may take place.

12.4.3 Multiple Conditions

The form of LOGICAL IF discussed above checks a single condition. But, in data processing, often a decision may have to be taken on multiple conditions (eg)

If (community is general and age is less than 18 and Marks is above 90%) or (community is backward and age is less than 20 and Mark is above 85%) etc.

Such multiple conditions are obtained by using LOGICAL OPERATORS between logical expressions. The list of logical operators and their meaning are given

OPERATOR	USAGE	MEANING
.AND.	exprn–1 .AND. exprn-2	True if both exprn–1 and exprn–2 are true.
.OR.	exprn–1 .OR. exprn–2	True if exprn–1 is true or exprn–2 is true or both are true.
.NOT.	.NOT. exprn.	True if exprn is false

Note that just as relational operators, the logical operators also have to be preceded and followed by a period (full-stop).

A multiconditional form of LOGICAL IF may be any of the following forms:

(1) IF ((A.GT.B).AND.(B.LT.C)). OR (G.LE.D). OR. (G.GE.A)

(2) IF.NOT.(A.EQ.B).AND.((B.EQ.D). OR. (D.LT.G))

Example 12.4 illustrates the use of multiconditional expressions.

EXAMPLE 12.4 100 candidates were tested for selection to Programmer's posts. An oral test and a written test were conducted and the percentage marks secured were entered in a floppy in the following format:

Character 1–3	IDENTIFICATION number
5–7	Candidate's name
8–10	% marks in written test
11–13	% marks in oral test

These records are to be proceessed and candidates getting 90% or more in one of the tests and 85% or more in the other will be selected and others rejected. The printed output should give the details on input record plus the result which should be either 'SELECTED' or 'REJECTED'. Also, the total number of candidates selected should be printed at the end. Write a program for the above job. Assume that an 80 character printer is used.

Solution:

A counter, say NREC, is used to keep track of records processed. It is initialised to zero before processing starts. It is incremented by 1 after processing each record. When it reaches 100, it indicates that processing of all records is over.

Another counter NSELN is initialised in the beginning and incremented whenever a candidate is found to be eligible for selection. The value of this is printed at the end.

Program

```
C           USE OF LOGICAL EXPRESSIONS
C           PRINT THE TITLES
            WRITE (6,5)
      5     FORMAT (21X, 'NMBR', 4X, 'NAME', 5X, 'WMRK', 4X, 'OMRK',
            5X, 'RESULT')
C           INITIALISE COUNTERS
            NREC=0
            NSELN=0
     10     READ (15,20) NMBR, NAME, MRKW, MRKO
     20     FORMAT (I3, A4, 2I3)
            WRITE (6,25) NMBR, NAME, MRKW, MRKO
     25     FORMAT (21 X, I3, 5X, A4, 2(5X, I3))
         0  IF((( MRKW. GE. 90). AND. (MRKO. GE. 80))
         1  .OR. ((MRKW. GE. 80) .AND. (MRK0. GE. 90)))
         2  GOTO 35
```

```
C      ELSE
       WRITE (6,30)
   30  FORMAT (‘+’, T55, ‘REJECTED’)
       GOTO 45
   35  WRITE (6,40)
   40  FORMAT (‘+’, T55, ‘SELECTED’)
       NSELN=NSELN+1
   45  NREC=NREC+1
       IF (NREC. LT. 100) GOTO 10
C      ELSE PRINT TOTAL NUMBER SELECTED
       WRITE (6,50), NSELN.
   50  FORMAT (31X, ‘TOTAL CANDIDATES
       SELECTED=’, I3)
       STOP
       END
```

In the above program, ‘+’ is used as control carriage instruction to print ‘SELECTED’ or ‘REJECTED’ in the same line as the other fields.

12.4.4 Logical Variables

In the above program, multiple conditions have been written as they are. This would be inconvenient if same set of conditions have to be written twice or more times or different combination of conditions have to be used in different statements. To overcome this inconvenience, each unique condition or group of conditions that always GOTOgether can be equated to a REAL variable and this variable can be used in the program statements.

(eg) IF ((A.OR.B). AND. C) GOTO 150. Such variables used in place of conditions are called LOGICAL VARIABLES.

To use such variables, a type statement should be given before any executable statement in the program. (eg)

LOGICAL A, B, C

Also the conditions each logical variable stands for should be defined in the beginning of the program (eg)

A = (MRKW.LT.80).AND(MRKO.GT.40)
B = (MAKW.GT.40).AND.(MRKO.LT.80)
C = (MAKW.GT.90).OR.(MRKO.GT.90)

The program under Example 12.5 illustrates the use of logical variables.

EXAMPLE 12.5 Write a program to read 4 REAL numbers in a card each of format F7.2 recorded with 1 blank in between. If the first number is greater than the last number and less than the sum of the last two numbers or if the first number is not less than the second, then the first number shuld be displayed on the screen. If the last number is equal to the sum of the squares of the second and third numbers or greater than or equal to the first number, then the last number should be displayed.

Program

```
C       |   | USE OF LOGICAL VARIABLES
        |   | LOGICAL A,B,C
        |   | A=(P.GT.S).AND.(P.LT.R+S)
        |   | B=(P.GE.Q)
        | 0 | C = (S.EQ.Q * Q + R * R).OR
        | 1 | (S.GE.P)
        |   | READ (5,10)P,Q,R,S
     10 |   | FORMAT (4(F7.2,1X)
        |   | IF (A.OR.B) GOTO
C       |   | ELSE
        |   | IF (.NOT.C) GOTO 30
        |   | ELSE
        |   | WRITE (6,15) S
     15 |   | FORMAT (6X,'S=',F7.2)
        |   | GOTO 30
     20 |   | WRITE (6,25) P
     25 |   | FORMAT (6X,'P=',F7.2)
     30 |   | STOP
        |   | END
```

Note that any LOGICAL variables will have 2 possible values only, namely TRUE or FALSE. The computer tests these conditions for branching.

12.4.5 Hierarchy of Logical Operations

Just as there is a hierarchy of Arithmetic operations, there is a hierarchy in logical operations also as listed below—

(1) First all arithmetic expressions contained in the logical expressions are computed.

(2) Next, all the relative operations are performed.

(3) Next all NOT operations are executed.

(4) Then all AND operations are executed.

(5) Finally, all OR operations are executed.

12.5 COMPUTED GOTO STATEMENT

The LOGICAL IF provides 2 alternatives and ARITHMETIC IF, 3 alternatives. The control statement which allows branching to more than three paths is called COMPUTED GOTO Statement. It is similar to CASE statement in PASCAL and ON-GOTO Statement in BASIC.

It is a special form of the GOTO Statement. It begins with the words Go To followed by a pair of parentheses containing any desired amount of statement numbers separated by commas. The right parenthesis is followed by a comma and a non-subscripted INTEGER variable. (eg) GOTO (10, 15, 30, 25, 20, 40, 50, 45), I

This statement means:

If value of I=1, GOTO the first mentioned statement (ie) 10, else, if the value of I=2, GOTO the second mentioned statement (ie) 15 and so on and if the value of I = 8 GOTO the eighth (last in this case), statement (ie) 45.

(ie) The computed GOTO is executed as follows: The computer will go next to the statement whose statement number appears in the parenthesised list in the order corresponding to the value of the INTEGER variable.

The list of statement numbers may be as long as desired and the same number may appear more than once in the list, if desired.

(eg) GOTO (5, 10, 5, 10, 5, 10), J

(If J = 1 or 3 or 5 GOTO 5 else if J = 2 or 4 or 6, GOTO 10)

In case the value of the INTEGER variable is other than the values covered by the COMPUTED GOTO statement, the control will GOTO the next sequential statement.

EXAMPLE 12.6 In an organisation, all the employees are assigned a 6 figure employee number. The numbers are so allotted that, for women employees, they end in 3 or 6 or 9 and for men-employees, the rest. The card format is

Columns	
1–6	Employees' Numbers
7–50	other data
51–80	blanks

Write a program to display on the screen the total number of men-employees and the total number of women-employees.

Program

```
C        |   | USE OF COMPUTED GOTO
         |   | NMEN=0
         |   | NWMEN=0
     8   |   | READ (5,10,END=25) NEMP
    10   |   | FORMAT (I6)
C        |   | ONLY EMPLOYEE NUMBER (NEMP) IS READ
C        |   | OTHER DATA FIELDS ARE IGNORED
C        |   | GET THE FINAL DIGIT N IN NEMP
         |   | I=NEMP/10
         |   | N=NEMP - 10 * I
         |   | GOTO (15, 15, 20, 15, 15, 20, 15, 15, 20), N
    15   |   | NMEN=NMEN+1
         |   | GOTO 8
    20   |   | NWMEN=NWMEN+1
         |   | GOTO 8
    25   |   | WRITE (0,30) NMEN, NWMEN
    30   | 0 | FORMAT (6X, ‘TOTAL MALE EMPLOYEES=’,
         | 1 | I6, 5X, ‘TOTAL FEMALE EMPLOYEES=’,
         | 2 | I6)
         |   | STOP
         |   | END
```

Note that, if N = 1, 2, 4, 5, 7 and 8, the control will GOTO statement 15 as directed in the COMPUTED GOTO statement. If N = 3, 6 or 9 it will GOTO Statement 20 again as indicated by COMPUTED GOTO Statement. If it is 0, it will GOTO the next sequential statement which happens to be the statement of number 15.

12.6 ASSIGNED GOTO STATEMENT

When a group of statements have to be executed repeatedly, but after different time of execution, the control has to change over to different statements, ASSIGNED GOTO is useful. It is similar to 'ALTER' statement in COBOL. It begins with the words "GOTO" followed by a non-subscripted INTEGER variable followed by a comma which is followed by a pair of parentheses containing any desired amount of statement numbers separated by commas. (eg) GOTO K, (25, 75, 100, 50)

This statement means GOTO Statement of number 25, or 75, or 100, or 50, whichever be the value most recently assigned to K.

K is called the 'BRANCH DESIGNATOR VARIABLE'. Before each execution of the ASSIGNED GOTO Statement, the Branch designator variable should be assigned different values. For this, an 'ASSIGN' statement is used.

(eg) ASSIGN 50 TO K.

The value assigned should be one of the labels (statement numbers) given within parentheses in the ASSIGNED GOTO statement.

EXAMPLE 12.7 The values of 3 INTEGER numbers, each of 3 digits, are given as input through the key-board. Write a program to compute their GCD and LCM and display them on the screen.

Solution:

(1) Let the 3 numbers be M, N and K

(2) Find G_1 = GCD of M and N

(3) Find L_1 = LCM of M and $N = M * N/G_1$

(4) Find G_2 = GCD of G_1 and K (G_2 = GCD of M, N and K)

(5) To calculate L_2 = LCM of L_1 and K, Compute G_3 = GCD of L_1 and K

(6) $L_2 = L_1 * K/G_3$ = LCM of M, N and K.

The procedure to find GCD is to be used thrice (ie) to find G_1, then G_2 and last G_3. But, after executing that procedure, the control goes to different steps at different times. This is accomplished by the use of ASSIGNED GOTO as illustrated below—

Program

```
C        TO FIND GCD AND LCM OF 3 INTEGER NUMBERS
C        USE OF ASSIGNED GOTO
         INTEGER GCD
         READ (0,5) M,N,K
    5    FORMAT (3I3)
C        SAVE VALUES
         I=M
         J=N
         ASSIGN 100 TO L
   10    IF (I - J) 15, 25, 20
   15    J=J - I
         GOTO 10
   20    I=I - J
         GOTO 10
   25    GOTO L, (100, 200, 300)
```

```
100   L1 = M * N/I
      J=K
      ASSIGN 200 TO L
      GOTO 10
200   GCD=I
      I=L1
      J=K
      ASSIGN 300 TO L
      GOTO 10
300   LCM=L1* K/I
      WRITE (0,305) GCD, LCM
305   FORMAT (6X, 'GCD=', I3, 5X,
      'LCM=', I9)
      STOP
      END
```

Summary

The computer normally executes the instructions in the sequence they are written in the program. To change this sequence, 'transfer of control' instructions have to be used. These are also called Branch or JUMP instructions.

GOTO and ASSIGNED GOTO are the unconditional branch instructions. 'GOTO' may be used to go back to an earlier statement or go ahead to a later statement in the program.

When a group of instructions is repeatedly executed but the control has to GOTO different statements on different occasions, ASSIGNED GOTO is used.

ARITHMETIC IF, LOGICAL IF and COMPUTED GOTO are the 'branch on condition' or conditional branch/jump instructions. Arithmetic IF provides for 3 alternatives, while logical IF provides for 2. COMPUTED GOTO is useful for multiple branching.

Exercise/Questions

12.1 Differentiate between the ASSIGNED GOTO and the COMPUTED GOTO statements in their format as well as application.

12.2 Assume that M, N, K are three distinct integer numbers. Write an Arithmetic IF statement to implement the following:

(a) If M is greater than both N and K, GOTO Statement number 15.

(b) If M is less than N but greater than K, GOTO Statement 20.

(c) If M is greater than N but less than K, GOTO Statement 25.

(d) If M is less than N as well as K, GOTO Statement 30.

12.3 Using Arithmetic IF Statement, write a program to print the roots of a Quadratic equation. Along with the roots, comment should be printed to indicate whether (i) roots are real and equal, or (ii) real and unequal, or (iii) imaginary.

12.4 Given four variables P, Q, R, S

Write LOGICAL IF statements to check which of them is equal to 50.0. If all of them are equal to 50.0 the control should GOTO statement 75, else to statement 100.

12.5 Write a program to compute the FACTORIAL of an integer number, using LOGICAL IF.

12.6 An organisation uses a 4 digit number as identification number of employee data records. For Class I employees, the first digit is 1. For Class II, it is 2 or 3; for Class III it is 4, 5 or 6, and for Class IV 7, 8 or 9. Write a program to read an indefinite number of employee records and print the number of employees in each category. Use COMPUTED GOTO.

12.7 Write a program to check whether a given INTEGER number is a PRIME (Prime number).

for NOTES

Chapter Thirteen

Loop Statements: DO Statement and Subscripted Variables and Processor Control Statements

13.1 INTRODUCTION

In program under Example 12.4 a counter NREC is used along with an LOGICAL IF Statement to check this counter to execute the program on 100 data records, say, in a loop. Also a statement is used to increment the counter after completion of execution of program on each record.

All these 3 operations can be automatically executed by the use of a LOOP Statement.

The Loop Statement in FORTRAN is better known as DO Statement.

13.1.1 DO Statement

A DO Statement is similar to PERFORM Statement in COBOL and for DO Statement in PASCAL. It is constructed in the following form: the word "DO" followed by a statement number which is followed in turn by an index definition.

eg,

$$\text{DO 10} \quad \underbrace{\text{I} = 1, 100, 2}_{\text{INDEX DEFINITION}}$$

The index definition contains an index variable, an equal sign, a lower limit, an upper limit, and an index increment which is optional. The index variable should be non-subscripted integer variable. I is the index variable in the above example, 1 is the lower limit, 100 the upper limit, and 2 index increment.

If the index increment is 1, it need not be mentioned.

eg, DO 20 J=1,100

This DO Statement tells the processor to repeatedly execute the successive sequence of statements up to and inclusive of statement numbered 20, starting from the index J=1, incrementing it by 1, up to the final index J=100.

The index limits (ie,) the lower index limit and the upper index limit may be integer constants as in above examples or integer variables.

eg,

```
DO 40 K=L, 500

DO 50 L=1, K

DO 100 M=I, J
```

The values of the variables used as index should have been computed previously elsewhere. Also both the lower and upper limits cannot be negative or 0. Also the upper limit should be always greater than the lower limit (ie,) index decrementation is not at all allowed in FORTRAN unlike in PASCAL.

Given below are some of the invalid forms of DO statements.

```
DO   20    A = 1,20
DO   50    I = -1,20
DO   60    J = -10,0
DO   70    K = 0,20
DO   80    L = -10,-1
DO   100   M = 200,100
```

Given below are some of the valid forms:

```
DO   15    I = 1,100
DO   25    J = 10,100
DO   35    K = 1,100,2
DO   40    L = 2,100,2
DO   50    M = 1,99,2
```

In the DO statement:

```
DO   50    M = 1,99,2
```

the processor will start with the very next statement to the DO statement and execute all the statements down to and including statement number 50, having the value of M as 1. When the statement 50 have been executed it will increment M by 2 (ie,) make it, 3 and go back to the statement next to the DO statement and repeat the above procedure. This goes on until M reaches 99.

In the statement:

DO 35 K=1, 100, 2, the index variable K will have value 1,3,5,7, ... etc... 99 and would never have the exact value of the upper limit 100. In such cases, the execution of the loop will be terminated after the index reaches the largest possible value without exceeding the upper limit (99 in the above example.)

The lower and upper index limits may be INTEGER constants or integer variables, as stated earlier, but can never be expressions involving mathematical operations.

```
DO   100   I = I + 2,15
DO   120   J = 1,M - 1
DO   140   J = M/N,M * N
```

are all invalid statements.

13.1.2 Subscripted Variables

In most cases, DO statements go with processing of subscripted variables. (refer Section 9.4.3)

DIMENSION. Statements have to be given at the beginning of the program while using subscribed variables (refer Art. 9.4.3.2).

EXAMPLE 13.1 A VECTOR of 50 real numbers is recorded on tape in blocks of 10 numbers each number of format F7.2. Write a program to print their sum.

```
C      USE OF DO STATEMENT
       DIMENSION A(50)
       READ (3,5) (A(I),I=1,50)
    5  FORMAT (10 F7.2)
       SUM=0.0
       DO 10 I=1, 50
   10  SUM=SUM+A(I)
       WRITE (6,15), SUM
   15  FORMAT (6X, 'SUM OF GIVEN ARRAY=', F9.2)
       STOP
       END
```

The statements under the DO statement up to and inclusive of the statement whose number is mentioned in the DO statement are said to constitute a 'DO LOOP'.

In the above example within the DO LOOP the index variable I is used in the statement: SUM=SUM+A(I). This facility is very useful in repeated computations of same type.

EXAMPLE 13.2 Given a vector of 100 elements of real numbers recorded in blocks of ten on magnetic tape, write a program to print it in the reverse order with 5 numbers per line. Each element is of format F8.3

Solution:

To reverse the given array, say, A, another array B should be created whose elements will be in the reverse order of elements in A.

```
C      REVERSING AN ARRAY
       DIMENSION A(100), B(100)
       READ (3,5) (A(I),I=1,100)
    5  FORMAT (10 F8.3)
       DO 10 I=1, 100
       J=100-I
   10  B(J)=A(I)
       WRITE (6,15) (B(J),J=1,100)
   15  FORMAT (6X, 5(F8.3, 5X))
       STOP
       END
```

The 'DO LOOP' in the Example 13.1 contains a single statement whereas in the Example 13.2 contains more than one. Note that DIMENSION statement has to be used both for input and output vectors (arrays).

13.1.3 Continue Statement

Any FORTRAN Statement can be used within a DO-LOOP but the loop should not end with a control statement (such as IF or GOTO). If it is necessary to end a DO-LOOP with a control statement;

a dummy statement, namely CONTINUE, should be used at the end of the loop. Naturally, the CONTINUE Statement should be assigned a statement number and the DO statement at the top of the loop should mention this number.

eg,

```
     |  |  DO 20 J=1, 50
     |  |  IF (A(I) - B(I)) 15, 20, 15
 15  |  |  ....................................
 20  |  |  CONTINUE
```

EXAMPLE 13.3 Write down a program to find the smallest element in the input vector A in Example 13.2, and display it on the screen.

Solution:

Let SMALL be the variable whose value is the smallest in the vector (array). Set it equal to A(I). This is used as a basis for comparison. Use an IF Statement under c. DO loop to compare the current value of SMALL with successive numbers in the vector A. When the loop operations end, SMALL will contain the desired value.

Program

```
C        |   | TO FIND SMALLEST ELEMENT IN A VECTOR
C        |   | IF STATEMENT IN DO LOOP
C        |   | USE OF CONTINUE STATEMENT
         |   | READ (3,5) (A(I),I=1,100)
       5 |   | FORMAT (10 F8.3)
         |   | SMALL=A(1)
         |   | DO 20 I=2, 100
         |   | IF (SMALL .LE. A(I)) GOTO 20
C        |   | ELSE
         |   | SMALL=A(I)
      20 |   | CONTINUE
         |   | WRITE (0,25), SMALL
      25 | 0 | FORMAT (6X, 'THE SMALLEST NUMBER
         | 1 | IN THE ARRAY=', F8.3)
         |   | STOP
         |   | END
```

The IF statement in the above program checks if SMALL is less-than or equal to A(I) in which case the control goes to CONTINUE statement so that the check operation may continue to compare SMALL with the next element in the array. Else, it makes SMALL equal to current value of A(I) and the control then goes to the CONTINUE-Statement to continue comparison of SMALL with the next element.

13.1.4 Use of IF to Leave a DO-LOOP

An IF statement or a branching statement may cause the computer to leave a DO-LOOP before the index has completely cycled in the normal manner. (refer Section 12.4.2.1.) The following example illustrates this.

EXAMPLE 13.4 A vector of 80 distinct numbers each of format F 5.2 are punched with 16 numbers per card. The value of a number B is punched in the same FORMAT in a single card which is placed before the cards containing the values of elements of A. Write a program to search for the element in A whose value equals B and display its index.

Solution:

The above is a problem on LINEAR SEARCH or SEQUENTIAL SEARCH. A linear search is done on an unsorted vector. In this search-method, the given vector is extensively checked to find whether it contains any element of given value. If the element searched is found, the loop operation will terminate prematurely. The value of index at the termination gives the position of the element searched for in the vector. If the element is not found at all, the search will end when the index is of upper limit. The condition indicated then is ambiguous.

It may be that the number searched for is the last element in the vector, or the search may be unsuccessful. To resolve this ambiguity the element searched for is placed as (N+1)th element in memory when N=size of array. In such cases, if the loop terminates when index equals (N+1) it means that the number searched was not in the original vector and hence the search is unsuccessful. Else, the index value at the terminating condition is returned as the position of the number in the vector.

Program

```
C           SEQUENTIAL SEARCH
            DIMENSION A(81)
C           DIMENSION IS SPECIFIED 1 EXTRA
C           TO ACCOMMODATE (N+1)TH ELEMENT
C           INPUT VALUE OF B FROM FIRST CARD
C           AND VALUES OF ELEMENTS OF VECTOR A
C           FROM SUBSEQUENT RECORDS
            READ (5,10) B,(A(I),I=1,80)
     10     FORMAT (F5.2/(16 F5.2))
            PLACE B AS 81ST ELEMENT OF VECTOR A
            A(81)=B
C           INITIATE SEARCH
            DO 20 I=1, 81
            IF (B. EQ. A(I)) GOTO 25
C           ELSE
     20     CONTINUE
     25     IF (I. EQ. 81) GOTO 35
C           ELSE
            WRITE (0,30) B,I
     30   0 FORMAT (6X, F5.2, ‘b̸ IS b̸’, I2, ‘TH b̸
          1 ELEMENT IN VECTOR A’)
            GOTO 45
     35     WRITE (0,40) B
     40   0 FORMAT (6X, F5.2, ‘b̸ IS NOT FOUND
          1 IN THE VECTOR A’)
     45     STOP
            END
```

In this example, the LOGICAL IF statement is used to go out of a DO-LOOP if the element searched is found before the upper index limit is reached. This is sometimes called ‘SPECIAL EXIT’ as compared to the exit from loop after upper index limit is reached, which is called NORMAL EXIT.

EXAMPLE 13.5 Illustrate the use of control statement (IF) for transfer of control within DO-LOOP.

EXAMPLE 13.6 Illustrates the use of control statement for possible transfer of control from the DO-LOOP to a range outside it. The GOTO statement may also be used in the above 2 cases. But transfer of control from outside the DO-LOOP to a statement within the loop is not at all allowed.

13.1.5 Nested DO-LOOPs

If a DO-LOOP exists completely within the range of another DO-LOOP, it is called a nested DO-LOOP and the situation is known as NESTING of DO-LOOPs.

eg,

```
     | |  DO 10 I = 1, 50                  )
     | |  DO 10 J = 1, 25  )               |
     | |  ...............  |               |  OUTER
     | |  ...............  |  INNER        |  LOOP
     | |  ...............  |  LOOP         |
     | |  ...............  |               |
  10 | |  ...............  )               )
```

In case of nested DO-LOOPs as above, the outer-loop is begun first and, during its first pass, it hands the control over to the inner-loop which will completely cycle itself and then give back the control to the outer-loop. The outer-loop will initiate the next pass and hands the control to inner-loop and so on. This process is continued up to the end of the upper index limit of the outer DO-LOOP.

In the above example the outer-loop will go through 50 passes and in each pass the inner-loop will be executed 25 times.

A maximum of 25 DO-LOOPs (inclusive of the outermost loop) can be had in a nest of DO-LOOP Of course most problems may require at the most 3 DO-LOOPs in a nest.

Also, the index-variables in NESTED-DO-loops must be different in name. The same index cannot be used for 2 DO-LOOPs if one is contained in the other. This precaution is not necessary in SEQUENTIAL DO-LOOPs (refer Example 13.8)

EXAMPLE 13.7 A vector K of 25 elements one punched on a card with 2 columns per element. Another vector L of 20 elements is punched on another card with 3 columns per element (Both K and L are intergers). Write a program to compute all the possible products of K and L and place the products in the vector M. The elements of M should be displayed on the screen with 10 elements per line.

Program

```
C      NESTED DO-LOOP
       DIMENSION K(25), L(20), M(500)
C      THE PRODUCT VECTOR M WILL HAVE 500 ELEMENTS
       READ (5,10) (K(I),I=1,25)
    10 FORMAT (25, I2)
       READ (5,15) (L(J),J=1, 20)
    15 FORMAT (20 I3)
       N=0
       DO 20 I=1, 25
       DO 20 J=1, 20
       N=N+1
```

```
 20     M(N) = K(I) * L(J)
        WRITE (0,25) (M(N),N=1,500)
 25     FORMAT (5X, 10(2X, I5))
        STOP
        END
```

The program under following example uses sequential DO-LOOPs.

EXAMPLE 13.8 For computing the area of a plot, a survey was made and 91 ordinates at intervals of 5 meters were taken covering the entire area.

These are recorded on tape with 13 values per block each value in meters of the format F7.3. Write a program to compute the area of the plot using Simpson's 1/3 rule and print it.

Solution:

AREA $\frac{D}{3}(y_1 + 4y_2 + 2y_3 + 4y_4 + 2y_5 + \cdots 2y_{89} + 4y_{90} + y_{91}$

where D=distance between ordinates
= 5 meters (in this case)

and $y_1, y_2, \ldots$ etc... y_{91} are the ordinates

Program

```
C             SEQUENTIAL DO-LOOPS
C             APPLICATION OF SIMPSON'S RULE
              DIMENSION y(91)
              READ (5,10) (y(I),l=1,91)
      10      FORMAT (13 F7.3)
              ODDSUM=0.0
              DO 15 I=3, 89, 2
      15      ODDSUM=ODDSUM+y(I)
C             ABOVE LOOP FINDS SUM OF ORDINATES
C             OF ODD SUBSCRIPTS y(3), y(5) ETC y(89)
              EVNSUM=0.0
              DO 20 I=2, 90.2
      20      EVNSUM=EVNSUM+y(I)
C             ABOVE LOOP FINDS SUM OF EVEN SUBSCRIPTED
C             ORDINATES y(2), y(4) ETC Y(90)
           0  AREA = 5.0/3 * (y(1) + 4.0 * EVNSUM
           1  +2.0 * ODDSUM + y(91))
              WRITE (6,25) AREA
      25      FORMAT (11X, 'AREA OF THE PLOT=', E 12.8, '␢ SQ.M')
              STOP
              END
```

In the above example, in both the DO-LOOPs the index incrementations is other than 1 (ie,) 2.

Also, the lower index is not 1. Though there are 2 DO-LOOPs, they do not form a nest since one is independent of the other. They are as such called SEQUENTIAL DO-LOOPs.

13.1.6 If Statement Under Nested DO-LOOP

In case of an IF Statement (Arithmetic IF or logical IF) being used in the innermost DO LOOP, the nest should end with CONTINUE Statement which should be numbered and this number should be mentioned in each of the DO-statements in the nest. Example 13.9 will illustrate this.

EXAMPLE 13.9 Write a program to print all the Pythagoras triplets between 1 and 50.

Solution:

Three integer numbers, say L, M, N, are said to constitute a PYTHAGORAS TRIPLET if $L^2 + M^2 = N^2$.

Since the values L, M and N are distinct integer-values, only combinations $L < M < N$ are tested. Also, since the computer can compose any number between 10^{-75} and 10^{75}, no input is required in this case.

```
C         | IF STATEMENT IN NESTED DO-LOOP
C         | USE OF CONTINUE STATEMENT
C         | PYTHAGORAS TRIPLETS
          | DO 10 L=1, 48
          | J=L+1
          | DO 10 M=J, 49
          | I=M+1
          | DO 10 N=I, 50
         0| IF ((L * L + M * M). NE. (N * N))
         1| GOTO 10
C         | ELSE (IF EQUAL)
          | WRITE (6,5) L,M,N
     5    | FORMAT (11X, 3 (5X, I2))
    10    | CONTINUE
          | STOP
          | END
```

The above program, incidentally, illutrates the use of outer-loop's index variable to fix the inner-loop's index limits (lower limit in this case). The program under the following Example 13.10 illustrats an occasion in which the outer-loop's index fixes or modifies the inner-loop's upper index.

EXAMPLE 13.10 Write a program to sort a vector of 100 integer numbers and print them in the ascending order. The numbers are recorded on tape with 10 number per block each of format I4. They are to be printed with 20 elements per line using an 132 character printer.

Solution:

The method adopted for sorting in this case is called BUBBLE SORT or INTERCHANGE SORT or SINKING SORT or SIFTING SORT.

This method of sorting involves $N - 1$ passes, when N is the total number of elements in the vector. In the first pass, there will be $N - 1$ comparisons. If the array is represented as M, M(1) is compared with M(2), them M(2), with M(3), and so on, interchanging the elements if they are not in the ascending order. Thus, by the end of Pass–I, the largest element would reach its proper position (ie) the last position. After each pass (in each pass the inner-loop is cycled completely once), the outer-loop's index is incremented by 1, indicating the next pass in which number of comparisons will be less by 1, than the previous pass, eg, in pass-2, there will be $N - 2$ comparisons only, since the largest element is already in the proper position. In the last and the $(N - 1)$th pass, there will be single comparison only.

On many occasions, the vector may get arranged in proper sequence before going through all the passes. In such cases, executing the other passes will be waste of computer time. To check whether the vector is already in order, a counter NEXCH is used. This is initialised to ZERO at the beginning of each pass. Whenever an exchange of 2 adjacent elements is performed, this counter is incremented by 1. If, at the end of a pass, its content is zero, it indicates that the array is already in order and exit is made from the loop.

Program

```
C       BUBBLE SORT
        DIMENSION M (100)
        READ (3,5) (M(I),I=1,100)
      5 FORMAT (10 I4)
        DO 20 I=1, 99
C       MODIFY UPPER INDEX OF INNER-LOOP
        K=100-I
        NEXCH=0
        DO 20 J=I, K
        IF (M(J). LE. M(J+1)) GOTO 20
C       ELSE EXCHANGE
        T=M(J)
        M(J)=M(J+1)
        M(J+1)=T
        NEXCH=NEXCH+1
        IF (NEXCH. EQ. 0) GOTO 25
C       ELSE
     20 CONTINUE
     25 WRITE (6,30) (M(I),I=1,100)
     30 FORMAT (4X, 20 (2X, I4))
        STOP
        END
```

The above program illustrates also that a 'special' exit can be made from a nest of DO-LOOPs also. (refer statement if (NEXCH.EQ.O) GOTO 25.

13.1.7 Flow Chart for DO-LOOPs

A HEXAGON is used in the flow-chart as the symbol for loop operations eg, refer Fig. 13.1.

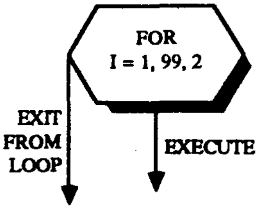


Fig 13.1

EXAMPLE 13.11 Draw a flow-chart to READ an vector of 100 elements of REAL mode and to print their average.

Solution:

Refer Fig. 13.2.

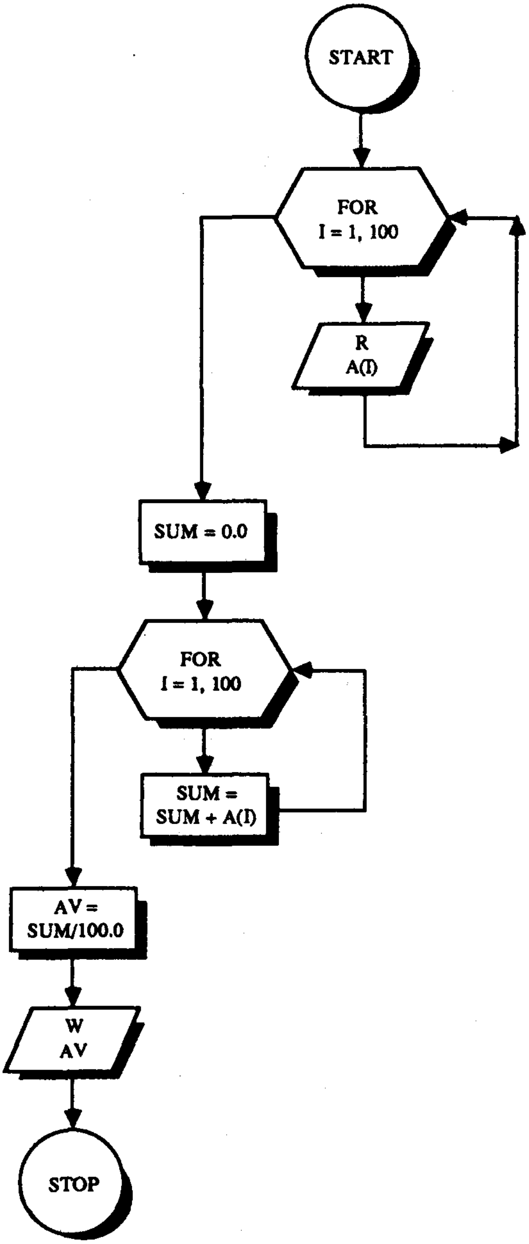

START
FOR
I = 1, 100
R
A(I)
SUM = 0.0
FOR
I = 1, 100
SUM =
SUM + A(I)
AV =
SUM/100.0
W
AV
STOP

Fig 13.2

13.1.8 Two Dimensioned Arrays

Nested DO-LOOPs are very useful for performing Arithmetic operations on multi-dimensioned arrays. Let us discuss this with respect to matrices (2-dimensioned arrays). Each element in a matrix is indicated by 2 subscripts.

eg, A(2,3);

The first subscript is called the 'ROW INDEX' and the second 'COLUMN INDEX'. A matrix of size 3 × 4 (3 rows by 4 columns) will be referred to as follows

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4
ROW 1	A (1,1)	A (1,2)	A (1,3)	A (1,4)
ROW 2	A (2,1)	A (2,2)	A (2,3)	A (2,4)
ROW 3	A (3,1)	A (3,2)	A (3,3)	A (3,4)

The matrix elements should be in the REAL mode while the subscripts are INTEGER constants or INTEGER variables.

eg,

A (2,3)
B (I,J)
C (K,6)
D (8,L)
E (N−1, N+1)
F (N+1,10)
G (15, N−1)
H (2 × K,15)

13.1.8.1 Dimension Statement for Multidimensional Arrays

Whenever matrices or 3 or more dimensioned arrays have to be taken as input or to be processed or to be given as output, DIMENSION Statements have to be written. For a matrix of 5 × 6 size, the dimension statement will be DIMENSION (5,6)

Similarly, if there is a three dimensioned array of size 6 × 6 × 6, the dimension statement will be DIMENSION (6,6,6).

13.1.8.2 Recording of Matrices

Matrices are entered with one row per record or line. If a matrix is of 4 rows, it will be punched in 4 cards, with 1 row per card or printed in 4 lines with 1 row per line. Each card or line or record will contain all the columns in a row. The format statement will, therefore, give the layout of 1 line or row.

13.1.8.3 I/O Statement of Matrix

The elements in a matrix can be read or written by self-indexing just as in the case of a vector. But the self-indexing in this case contains 2 index definitions, one for the row and the other for column.

eg, READ $(0,5)((A(I,J),J = 1,5),I = 1,4)$

will read a matrix of 4 rows and 5 columns. It will take the first row, read the elements in columns 1 to 5, one by one, then take the second row, repeat the process and so on.

Similarly the Statement

WRITE $(6,10)((B(K,L),L = 1,4),K = 1,5)$ will print a matrix of 5 rows and 4 columns with one row per line.

EXAMPLE 13.12 Matrix A is of size 4×5 and its elements are recorded in 4 cards (1 card/row) each of format 5 F7.3.

Matrix B is of size 5 × 6 and its elements are recorded in 5 cards each of format 6 F7.3.

Write a program to print the Matrix P which is the product of A and B.

Program

```
C          TWO DIMENSIONED ARRAYS
C          MATRIX MULTIPLICATION
           DIMENSION A (4, 5), B(5, 6), P(4, 6)
           READ (5,10) ((A(I,J),J=1,5),I=1,4)
      10   FORMAT (5 F7.3)
           READ ( 5,15) ((B(J,K),K=1,6),J=1,5)
      15   FORMAT (6 F7.3)
           DO 20 I=1,4
           DO 20 K=1,6
           P (I,K)=0.0
           DO 20 J=1,5
      20   P(I,K)=P(I,K)+A(I,J)*B(J,K)
           WRITE (6,25) ((P(I,K),K=1,6),I=1,4)
      25   FORMAT (5X, 6(2X, F10.3))
           STOP
           END
```

Note that the innermost loop computes each element in the matrix P, the intermediate loop each row of 6 columns and the outermost loop the 4 rows in P.

13.2 PROCESSOR CONTROL STATEMENTS

The processor control statements are (1) STOP, (2) PAUSE, (3) CALL, (4) RETURN, and (5) END statements.

13.2.1 Stop Statement

This is sometimes called 'Halting Control Statement'. This is an executable statement. When the 'STOP' statement is come across, the computer is brought to a halt and has to be restarted beginning all over again. As shown in all the programs discussed so far, the 'STOP' statement is used at the end of each program followed by the END statement.

13.2.2 Pause Statement

This statement will cause the computer to 'stop temporarily', (ie,) when the PAUSE statement is come across, the computer will halt, but if the START button/switch on the console is pressed, the computer will continue with the statement next to the PAUSE statement in the program.

In addition to causing the computer to halt, the PAUSE statement will also cause the message 'PAUSE 00000' to be displayed on the console typewriter or screen or on the printer.

PAUSE statement is useful in making provision in the user's program to halt the computer in case of errors occurring during execution of Arithmetic operations. In such cases, more than one PAUSE statement may have to be used in the program. In such cases, it would be desirable to know the error that caused the PAUSE or rather the PAUSE statement that has halted the computer. For this purpose, the programmer can write the word 'PAUSE' followed by an integer number of maximum 5 digits.

eg,

PAUSE 55555

When the computer halts due to this PAUSE statement, it will display 'PAUSE 55555' on the screen or printer etc. It is not necessary that this integer number used to identify the causes of "PAUSE" should be of 5 digits. If it is of less number of digits, the computer will supply the requisite number of leading zeros to make the displayed number to five digits.

EXAMPLE 13.13 There are indefinite number of cards each of format as follows

Characters 1–7	A (F7.3)
8–14	B (F7.3)
15–21	C (F7.3)
22–80	BLANKS

The cards are to be processed to compute D = $\sqrt{A} + B/C$ in each case. Provision should be made to check whether in any card A is negative, if so, the computer should be made to halt and print "PAUSE 00033". Also it should be checked whether C is 0 and if so the computer should be made to halt and print "PAUSE 00222". If both the above error conditions exist, the processor should be made to halt and print "PAUSE 04444". If only no error condition exists; D should be conputed and the values of A, B, C and D are to be printed.

Program

```
C          |   | USE OF PAUSE STATEMENT
        5  |   | READ (5,10, END=40) A,B,C
       10  |   | FORMAT (3 F7.3)
           |   | IF (A.GE.0) GOTO 25
C          |   | ELSE
           |   | IF (C. NE. 0) GOTO 15
C          |   | IF C=0 AND A IS NEGATIVE
           |   | PAUSE 4444
           |   | GOTO 5
C          |   | IF C ≠ 0 AND A is NEGATIVE
       15  |   | PAUSE 33
           |   | GOTO 5
       25  |   | IF (C. NE. 0) GOTO 30
C          |   | IF A IS NOT NEGATIVE BUT C=0
           |   | PAUSE 222
           |   | GOTO 5
C          |   | IF NO ERROR CONDITION
       30  |   | D=SQRT (A)+B/C
           |   | WRITE (6,35) A,B,C,D
       35  | 0 | FORMAT (6X, 'A=', F7.3, 5X, 'B=',
           | 1 | F7.3, 5X, 'C=', F7.2
           | 2 | 5X, 'D=', F7.2)
           |   | GOTO 5
       40  |   | STOP
           |   | END
```

Note that after each PAUSE statement a control statement (GOTO 5 in this case) is to be provided so that, when, after a 'PAUSE' condition, the start button of the console is pressed (ie,) if the computer is restarted, it should continue the processing with other records.

13.2.3 Call and Return Statements

A 'CALL' instruction is used to make the computer branch to a SUBROUTINE SUBPROGRAM and execute it.

A 'RETURN' instruction is used for the computer to branch back to the main program after executing a sub-program.

These are discussed in the next chapter.

13.2.4 End Statement

The END statement should be the last written statement in the program or subprogram. It is not an executable statement. Its purpose is to tell the computer during compilations that the end of the source program or subprogram has been reached.

Summary

The DO-LOOP is very useful in executing one or more steps repeatedly.

The major application of DO-LOOP is in processing vectors and arrays. The element in a vector is represented as a subscripted variable.

When vectors or arrays are processed, suitable DIMENSION STATEMENTS has to be provided in the beginning of the program.

If a control statement occurs within a DO-LOOP, the loop has to end with a CONTINUE statement. (some authors advise ending every DO-LOOP with a CONTINUE statement. This is not necessary).

A DO-LOOP within another DO-LOOP is called NESTED DO-LOOP. Nested DO-LOOPs are useful in operations like sorting vectors and processing of multi-dimensioned arrays like matrices.

STOP, PAUSE, END, CALL and RETURN are processor (computer) control statements. STOP is used to halt the computer after processing of all records is completed. PAUSE may be used to halt the computer in-between, on some error condition. CALL is used to make the processor branch to a SUBROUTINE subprogram and RETURN is used to make it branch back to the main program from any subprogram.

END statement is to be used at the end of any program or subprogram.

Exercise/Questions

13.1 Write down the algorithm for (1) Sequential search, (2) Bubble sort, (3) Finding smallest number in the given vector, and (4) Matrix multiplication.

13.2 Draw flow-charts for the above problems.

13.3 Write a program to add the 2 matrices A and B of size 5 × 4.

13.4 Write a program to subtract matrix B from matrix A in the above problem.

13.5 Write a program to arrange the elements in the vector given under Example 13.10 in descending order.

13.6 Write a program to find the largest element in the vector given in Example 13.3.

13.7 Write a DO-LOOP to count all numbers greater than 15.55 in a 200 element Array–A.

13.8 Write a program to print all the prime numbers up to 1000.

13.9 Rewrite the program under Example 13.8 using trapezoidal rule to compute the area.

13.10 Write a program using nested DO-LOOP to print multiplication table up to 10.

for NOTES

Chapter Fourteen

Sub-programs (Sub-routines) and Specification Statements

14.1 INTRODUCTION

In many of the programs, the same sequence of computations may have to be performed repeatedly, eg, finding VARIANCE of two or more vectors for SQC.

Writing the corresponding sequence of statements, every time these computations have to be done, will be time-consuming and also will result in waste of memory-space. As such, program modules called SUB PROGRAMS incorporating the sequence of instructions (those are to be repeatedly executed) are prepared, compiled and stored in the memory. The user's program can use these sub-programs whenever necessary. In case of big computers, different users can use the same sub-program. (It need not be written along with each user's programme).

14.1.1 Types of Sub-Programs

The two types of sub-programs are (i) OPEN sub-program, and (ii) CLOSED sub program.

The BUILT-IN or LIBRARY FUNCTIONS discussed in Chapter 12 are the open sub-Programs.

14.1.2 Closed Sub-Programs

In many cases, besides the library functions, certain sequence of computations may have to be repeatedly used, eg, calculating factorial to sum up series: $1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} \cdots$

In such cases, the user can prepare a suitable sub-programm and use it repeatedly in his program. The user-prepared-sub-programs are called CLOSED sub-programs. The closed sub-programs become essential only when no library function is available for the job.

FORTRAN provides 3 types of closed sub-programs—

(i) Arithmetic Statement Function;

(ii) Function Sub-Program; and

(iii) Sub-routine Sub-Program.

14.1.3 Arithmetic Statement Function

An Arithmetic Statement function is a sub-program containing a single statement giving a single result.

eg,

$$\mathrm{SUM(A, B, C) = A + B + C}$$

Such a statement is called the Function Defining Statement. It contains a selected name (SUM in this case), followed by some variables separated by commas within parentheses. The variables are called Arguments or Dummy variables. These are followed by an assignment sign (=) and a definition of functional relationship, which defines the operations to be performed (A+B+C, in this case). The function definition should contain the same set of variables on both sides of '=' sign. The selected name (SUM in this case) is called FUNCTION NAME. Selection or assignment of a name to a function is done by the programmer. The function name should be according to FORTRAN naming rules, (ie,) 1 to 6 alphameric characters with the first character being an alphabet. The mode of the function name should be the same as that of Arguments.

eg, ROOT (A,B,C,D)

IROOT (M,N,K,L)

Subscripted variables should not be used in the expression of a function defining statement..

eg,

$\mathrm{CALCN(X, Y) = (X(I) * Y(I))/(Y(I) - X(I))}$ is not a valid function defining statement.

The 'dummy arguments' listed in the function defining statement are used to indicate how the values of the actual variables in the program are to be operated (manipulated).

For example, using the Arithmetic statement function SUM (A,B,C)=A+B+C, the program may contain a statement

$$\mathrm{P = SUM(Q, R, S) * SUM(X, Y, Z)}$$

On coming across the term SUM, the computer will search for a function of that name, substitute the values of actual variables in place of the dummy assignments, perform the specified operations, compute the result and return it to the main program. These operations are performed every time the function name is come across in the program.

Since the function defining statement is to be referred to whenever the function name appears in the program, it should precede all the executable statements in the program.

EXAMPLE 14.1 To compute the height H of a tower at a distance, the angles subtended by its top, say θ_1 and θ_2, are measured from 2 points P_1 and P_2. The values of θ_1, θ_2 in degrees and the distance D between P_1 and P_2 in meters are recorded each of format F7.3 on a card. Write a programe to compute the height. Use an Arithmetic statement function to conpute $\tan\theta$.

Solution:

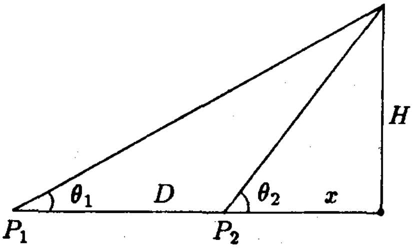


Fig 14.1

$$\left.\begin{aligned}
\tan\theta_2 &= \frac{\mathrm{H}}{\mathrm{x}}\\
\mathrm{x} &= \mathrm{H}/\tan\theta_2\\
\tan\theta_1 &= \frac{\mathrm{H}}{\mathrm{D}+\mathrm{x}}\\
(\mathrm{D}+\mathrm{x})\tan\theta_1 &= \mathrm{H}\\
\left(\mathrm{D}+\frac{\mathrm{H}}{\tan\theta_2}\right)\tan\theta_1 &= \mathrm{H}
\end{aligned}\right\}
\left\{\begin{aligned}
&\left(\frac{\mathrm{D}\tan\theta_2+\mathrm{H}}{\tan\theta_2}\right)\tan\theta_1 = \mathrm{H}\\
&\frac{\mathrm{D}\tan\theta_2\tan\theta_1+\mathrm{H}\tan\theta_1}{\tan\theta_2} = \mathrm{H}\\
&\mathrm{D}\tan\theta_2\tan\theta_1+\mathrm{H}\tan\theta_1 = \mathrm{H}\tan\theta_2\\
&\mathrm{D}\tan\theta_2\tan\theta_1 = \mathrm{H}(\tan\theta_2-\tan\theta_1)\\
&\mathrm{H} = \frac{\mathrm{D}\tan\theta_1\tan\theta_2}{\tan\theta_2-\tan\theta_1}
\end{aligned}\right.$$

Program

```
C       TO COMPUTE THE HEIGHT OF A TOWER
C       USE OF ARITHMETIC STATEMENT FUNCTION
     0  TAN(THTA)=(SIN(THTA/57.29578)/
     1  COS(THTA/57.29578)
C       MAIN PROGRAM FOLLOWS
        READ (5,5), THTA1, THTA2, D
5       FORMAT (3F7.3)
        T1=TAN(THTA1)
        T2=TAN(THTA2)
        H=D * T1 * T2/(T2 - T1)
        WRITE (6,10)H
10      FORMAT (6x, ‘THE HEIGHT OF TOWER=’, F10.4, ‘b̸ METERS’)
        STOP
        END
```

Note that THETA is divided by 57.29578 to convert degrees into radians.

14.1.4 Function Sub-program

If a needed function is neither a library function nor can be expressed in a single statement, Function Sub-Program has to be used.

A function Sub-program accepts values of one or more arguments from the main program and returns to it a single resultant value of the function which it computes.

The Function Sub-program is by itself a complete program. It is written separately from the ‘main’ program in which it is used. The Function Sub-program begins with a title line containing the word FUNCTION followed by the name of a function and a list of dummy arguments, separated by commas within parentheses.

eg,

Function DEFN (A,B,C,D)

The function name should be according to FORTRAN naming rules and should be in the same mode as the 'significant variables' in the dummy arguments. For example, if the dummy arguments are (X,Y,Z,I) where I stands for subscript, the Function name should agree with the mode of X,Y,Z (ie,), REAL mode.

The mode of function can be explicitly specified also, eg,

REAL FUNCTION INDENT (X,Y,Z)

This declares the function name INDENT as REAL.

Following the title line, the statements in the sub-program are written indicating the procedure to evaluate the value of function. (The result is the same as the function or the name of result is the same as that of the function). This is stated as, the FUNCTION has a single value for its result which becomes the value of the FUNCTION name in the expression in which the name is used.

The FUNCTION sub-program ends with a RETURN and an END statements.

A Function Sub-Program is 'Invoked' in a program by its name followed by the list of actual variables (corresponding to the dummy variables)

eg,

If the Function Sub-Program is titled FUNCTION POLAR (A,B,C), it will be invoked as, say,

Z=POLAR (W,X,Y)

On this the processor will branch from the main program to the sub-program of identification POLAR, substitute the values of the actual variables W,X,Y in place of dummy arguments A,B,C, respectively (with the same sequence as they are mentioned in the respective statements), and compute the result.

The RETURN Statement in the sub-program makes the computer go back to the 'main' program and assign the computed result to Z and then 'pick up' where it left off in the main program.

EXAMPLE 14.2 Write a FUNCTION-SUB-PROGRAM to compute GCD of any 2 integer numbers and use it to compute the GCD of 4 integer numbers to be input through Key-board. Assume each number is of maximum 3 digits. The result is to be displayed on the screen.

Solution:

```
C           USE OF FUNCTION SUB-PROGRAM
C           FUNCTION TO COMPUTE GCD
            INTEGER FUNCTION GCD (M,N)
      5     IF(M-N) 10, 20, 15
     10     N=N-M
            GOTO 5
     15     M=M-N
            GOTO 5
            GCD=M
            RETURN
            END
```

```
C          MAIN PROGRAM FOLLOWS
           INTEGER G1,G2,G3
           READ (O,20), I,J,K,L
      20   FORMAT (4I3)
C          COMPUTE G1=GCD OF I AND J
           G1=GCD(I,J)
C          COMPUTE G2=GCD OF G1 AND K
           G2=GCD(G1, K)
C          COMPUTE G3=GCD OF G2 AND L
C          G3 IS GCD OF I,J,K,L
           G3=GCD(G2, L)
           WRITE(0,25) G3
           FORMAT (6x, 'GCD OF GIVEN 4 NUMBERS=', I3)
           STOP
           END
```

Note that the Function Sub-program GCD is used thrice in the program. Also, the mode of function is explicitly specified.

14.1.4.1 Multiple Return Statements

The sub-programs are stored separate from the 'main' programs in the memory. When the sub-program is to be executed, with the help of the Function name the sub-program is accessed. Thus, the Function name serves as ENTRY POINT to the sub-program. Similarly the RETURN statement serves as the EXIT point. In case the result in the sub-program will be of different computations and control statements are used to branch to different computations, there should be as many RETURN Statements in the sub-program as the different computations. Example 14.3 will illustrate this.

EXAMPLE 14.3 Write a Function sub-program to compute the factorial of a positive integer number and use it to sum the series; $1 + \frac{x}{1!} + \frac{x_2}{2!} + \frac{x_3}{3!} + \cdots$

The value of x is to be given through the key-board in the format F 7.5. The summing is to be performed until the term becomes less than 0.00001. The sum should be displayed on the screen. Also, draw the flow-chart for the sub-program as well as for the 'main' program.

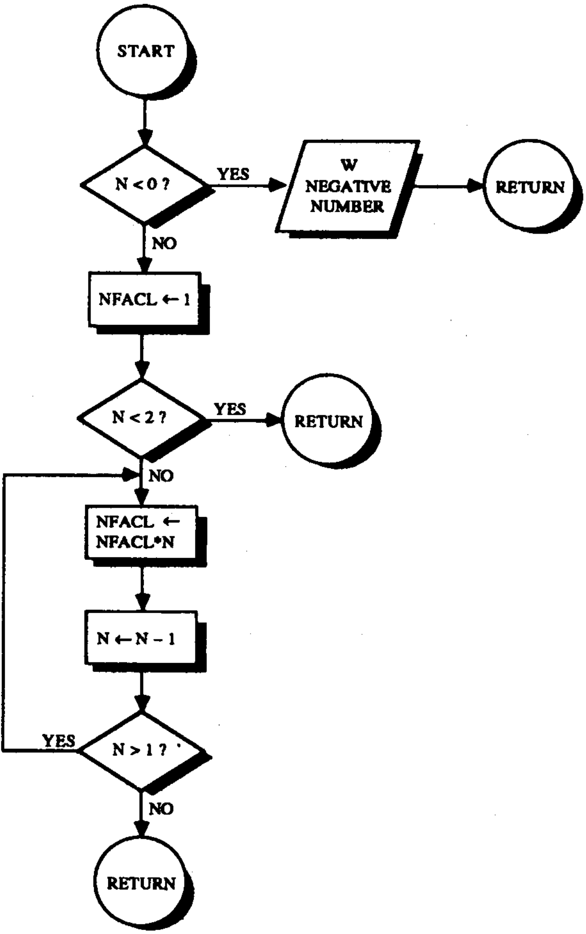


Fig 14.2

Solution:

Flow Chart for Sub-program (refer Fig. 14.2).

Flow Chart of main program (refer Fig. 14.3)

Program

```
C        FUNCTION SUB-PROGRAM TO COMPUTE FACTORIAL.
         FUNCTION NFACL(N)
         IF (N, GE. 0) GOTO 10
C        ELSE IF NEGATIVE
         WRITE (0,5)
      5  FORMAT (6x, 'NEGATIVE NUMBER')
         RETURN
     10  NFACL=1
         IF (N, GE. 2) GOTO 15
C        ELSE IF N=1 OR 0
         RETURN
     15  NFACL=NFACL*N
         N=N-1
         IF (N. GE.2) GOTO 15
C        ELSE
         RETURN
         END
C        MAIN PROGRAM FOLLOWS
         READ (0,20) X
     20  FORMAT (F7.5)
         SUM=1.0
         ANUMR=1.0
         I=0
C        COMPUTE THE NUMERATOR
     25  ANUMR=ANUMR*X
C        COMPUTE THE DENOMINATOR
         I=I+1
         DNMR=FLOAT (NFACL(I))
C        COMPUTE THE TERM
         TERM=ANMR/DNMA
C        CHECK IF TERM<0.00001
         IF (TERM. LT. 0.00001) GOTO 30
C        ELSE ADD TERM TO SUM
         SUM=SUM+TERM
         GOTO 25
     30  WRITE (0,35) SUM
     35  FORMAT (6x, 'THE SUM OF GIVEN
         SERIES=', F10.3)
         STOP
         END
```

Note that though there are 2 or more RETURN Statements in the sub-program, there one END Statement for it.

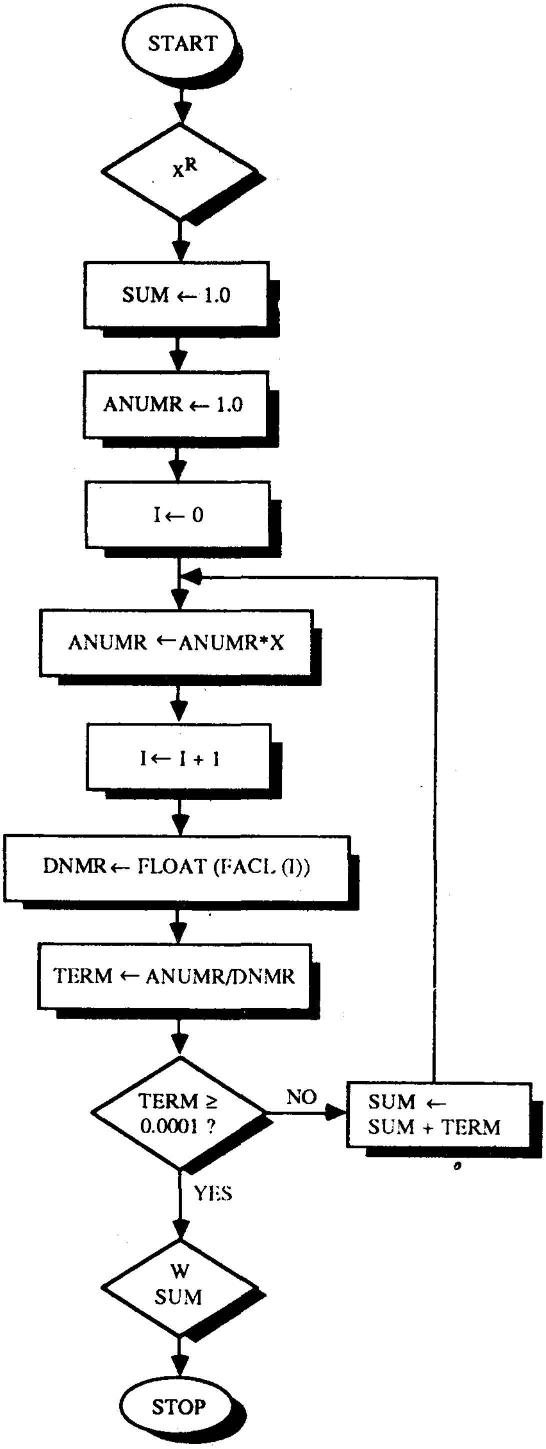

START
X^R
SUM ← 1.0
ANUMR ← 1.0
I ← 0
ANUMR ← ANUMR*X
I ← I + 1
DNMR ← FLOAT (FACL (I))
TERM ← ANUMR/DNMR
TERM ≥ 0.0001 ?
NO
SUM ← SUM + TERM
YES
W SUM
STOP

Fig 14.3

14.1.4.2 Arrays in Sub-Programs

If a FUNCTION Sub-Program has a dummy argument which is a vector (array), a DIMENSION statement should be written containing the name of that argument. This is essential to facilitate the use of subscripted variable in the 'main' program. The array size in the calling (main) program may be different from that in the sub-program and may also be different from run to run. So the size of the array in the sub-program is chosen as large as the maximum size that may be needed.

EXAMPLE 14.4 A vector P has 80 elements recorded on tape in blocks of 10 elements, each element of format F7.2.

Another vector Q also has 80 elements recorded on tape in blocks of 8 elements, each element of format F8.3.

Use a Function Sub-Program to compute the SUM of each array and check which has greater SUM and display the message on the screen.

Program

```
C       FUNCTION SUB-PROGRAM FOR
C       SUMMING AN ARRAY
        FUNCTION SUM(A, N)
        DIMENSION A(100)
        SUM=0.0
        DO 5 I=1, N
     5  SUM=SUM+A(I)
        RETURN
        END
C       MAIN PROGRAM FOLLOWS
        DIMENSION P(80), Q(80)
        READ (3,10) (P(I),I=1,80)
    10  FORMAT (10 F7.2)
C       COMPUTE SUM OF ARRAY P
        SUMP=SUM(P,80)
        READ (3,15) (Q(J),J=1,80)
    15  FORMAT(8 F8.3)
C       COMPUTE SUM OF ARRAY Q
        SUMQ=SUM(Q,80)
        IF (SUM P-SUMQ) 20,30,40
    20  WRITE (0,25)
    25  FORMAT (6x, 'SUM OF ARRAY Q IS GREATER')
        GOTO 50
    30  WRITE (0,35)
    35  FORMAT (6x, 'BOTH ARRAYS HAVE SAME SUM')
        GOTO 50
    40  WRITE (0,45)
    45  FORMAT (6x, 'SUM OF ARRAY P IS GREATER')
    50  STOP
        END
```

Note: (1) In the Example 14.2, The FUNCTION name is defined explicitly whereas in Examples 14.3 and 14.4, it is defined implicitly. (2) In 14.4 the FUNCTION name is in the same mode as the significant variable A, and not with the index variable I. (3) In 14.4, dimension statements are used both in the sub-program and the 'main' program. (4) While computing SUMP: the values of variable

P that are read from the input device are substituted in place of A in the sub-program. Similarly the values of B are substituted while computing the SUMQ. This substitution is called PARAMETER PASSING.

The method of passing of the value of variable read as input, to the sub-program is called PASSING-AS-VARIABLE. The corresponding dummy variable is called VARIABLE PARAMETER. In Example 14.4, A is a VARIABLE PARAMETER. So also M and N in Example 14.2 and N in 14.4.

In Example 14.4, the value of N in the sub-program is substituted by 80 while calculating the sums. This method of substituting a number as such from the main program in place of a variable in the sub-program is called PARAMETER PASSING-BY-VALUE. The corresponding variable (N in this case) is called VALUE PARAMETER.

14.1.5 Control of Memory Allocation

As already discussed, the data and the user's program (in machine code) are to be stored in the memory of the computer (stored program concept) for execution. Every variable used in the program will be allocated separate memory locations. A programmer should always aim at using minimum possible area in the memory for his data and program. The EQUIVALENCE and COMMON statements are provided in FORTRAN to give the programmer some control over the allocations of memory locations for the variables. These statements come under the category of SPECIFICATION statements.

14.1.5.1 EQUIVALENCE Statement

The EQUIVALENCE statement allows the programmer (user) to assign different variables to the same storage location. These variables should occur in the same program or in the same sub-program. But it should be made sure that the varaibles sharing the same loctions are never needed to be used at the same time.

The form of the EQUIVALENCE statement consists of the word EQUIVALENCE followed by sets of parentheses that contain names of any number of variables separated by commas. The variables are those that are to be assigned the same memory locations. The variables that are to be assigned same memory location should be in the same mode, eg, EQUIVALENCE (A,B) and EQUIVALENCE (I,J) are both valid.

But EQUIVALENCE (A,I) and EQUIVALENCE (J,B) are invalid.

If there are 2 or more sets of variables, each set to be assigned the same location, it is not necessary to write separate EQUIVALENCE statements. They can be written along with a single EQUIVALENCE statement with each set being enclosed in separate parentheses and a comma to separate the parentheses, eg,

EQUIVALENCE (A,B,C), (I,J)

This means 'Assign the same memory location to A, B and C and another location to I and J'.

The variables listed in an EQUIVALENCE statement may be subscripted. But the subscript should be a constant and not a variable.

EQUIVALENCE (A(1), B(12)) is valid
But EQUIVALENCE (A(K), B(10))
or EQUIVALENCE (A(5), B(L))
or EQUIVALENCE (A(M), B(N)) will be invalid.

14.1.5.2 Common Statement

Common Statement has to be used in case variables in main program and variables in the sub-program/s can share the same storage locations. In such cases, there should be a COMMON Statement one each in MAIN PROGRAM and SUB PROGRAM. eg,

(in main program)

```
COMMON A, B, I
```

and (in sub-program)

```
COMMON P, Q, J
```

These mean A will share the same location as P, B same as Q and I same as J. Note: variables to share the same location should be in the same mode.

As given above, a COMMON statement has to be written with the word COMMON followed by the names of variables separated by commas. The variables should not be enclosed in parentheses and there should be a blank or a few blanks after the word COMMON and not a comma.

Incidentally, it should be noted that there is no restriction as to the number of programs that can have COMMON statement. Program under Example 14.5 illustrates the use of COMMON and EQUIVALENCE statements.

EXAMPLE 14.5 The areas of two plots A and B are to be compared. The data regarding A are contained in a deck of cards. The first card in the deck gives the value of D in meters and the value of N. D is the equal distance in meters between ordinates and N the number of ordinates. D is of format F4.1 and N, I3. The other cards contain value of N ordinates $V_1, V_2, \ldots$ etc... V_n with 10 values per card each of format F7.3.

The data regarding B are contained in another deck of cards. The first card in this deck also gives the distance between ordinates H in meters and the number of ordinates M. This card is of same format as the first card in the other deck. The values of M ordinates (z) are on other cards in the deck, each card of format 10 F7.3. The ordinates are in meters in both cases. Write a program to find which of the 2 plots is of greater area. Use a sub-program to compute the area using trapezoidal rule. Use COMMON and EQUIVALENCE statements wherever possible.

Solution:

Trapezoidal Rule: If y_1, y_2 etc... y_n are ordinates taken at a distance W,

$$\text{AREA} = \frac{W}{2}(y_1 + 2y_2 + 2y_3 \cdots + 2y_{n-1} + y_n)$$

Program

```
C     USE OF COMMON AND EQUIVALENCE STATEMENTS
C     FUNCTION SUB-PROGRAM TO COMPUTE AREA
C     BY USE OF TRAPEZOIDAL RULE
      FUNCTION AREA (Y,X,L)
      DIMENSION Y(500)
      COMMON Y,X,L
      SUM=0.0
      K=L-1
      Do 5 I=2,K
```

```
    5     SUM=SUM+Y(I)
          AREA=X/2.0*(Y(1)+2.0*SUM+Y(L))
          RETURN
          END
C         MAIN PROGRAM FOLLOWS
          DIMENSION V(500), Z(500)
          COMMON V, D, N
          EQUIVALENCE (V,Z), (D,H), (N,M)
          READ (5,10), D, N, (V(J),J=1, N)
   10     FORMAT (F4.1, I3/(10F7.3))
          A1=AREA (V, D, N)
          READ(5,10), H,M(Z(J),J=1,M)
          A2=AREA (Z, H, M)
          WRITE (6,20) A1, A2
   20     FORMAT (6x, 'A1=', E 14.8, 5x,
          'A2=', E 14.8)
          IF (A1 - A2) 25, 45, 35
   25     WRITE (6,30)
   30     FORMAT ('+', T-50, 'A2 IS BIGGER')
          GOTO 55
   35     WRITE (6,40)
   40     FORMAT ('+', T50, 'A1 IS BIGGER')
          GOTO 55
   45     WRITE (6,50)
   50     FORMAT ('+', T50, 'BOTH A1 and A2 ARE OF EQUAL AREA')
   55     STOP
          END
```

The variables Y,X,L in the sub-program have corresponding arguments V,D,N and Z,H,M in the main program. So Y,X,L are called GLOBAL variables.

For the index variable, I is the sub-program, there is no corresponding variable appearing in the main program. Such a variable is called LOCAL VARIABLE.

14.1.6 Subroutine Sub-programs

A SUB-ROUTINE SUB-PROGRAM can campute and communicate to the main program, values of one or more variables (of course, for returning a value of a single variable, FUNCTION sub-program is preferable)

The SUB-ROUTINE SUB-PROGRAM begins with a title SUB-ROUTINE followed by the name of the Sub-routine. The rules for framing the name of a Sub-routine is the same as those of the Function; but there is no mode associated with it. As in FUNCTION SUB-PROGRAM this name is followed by DUMMY ARGUMENTS separated by commas, within parentheses.

eg,

```
SUB-ROUTINE TRAN (M,N,K)
SUB-ROUTINE SENS (A,B,C,D)
SUB-ROUTINE IJAC (I,J,K)
SUB-ROUTINE JAM (P,Q,R)
SUB-ROUTINE MATH (P,Q,S,I)
SUB-ROUTINE DIV (X,Y,J)
```

In normal practice, the SUB-ROUTINE is given a meaningful name, eg, if a SUB-ROUTINE is prepared to transpose a matrix of size M × N, the title may be SUB-ROUTINE TRANSPS (A,M,N) where A represents the element of the matrix and the name TRNSPS indicates 'TRANSPOSE'. Just as in the FUNCTION SUB-PROGRAM the SUB-ROUTINE SUB-PROGRAM should end with RETURN and End statements. DIMENSION statements should be used in the sub-program if arrays are involved. COMMON and EQUIVALENCE statements may be made use of wherever possible.

The main differences between the FUNCTION SUB-PROGRAM and the SUB-ROUTINE SUB-PROGRAM are (1) The FUNCTION SUB PROGRAM can return only one result whereas the SUB-ROUTINE SUB PROGRAM can return one or more results. (2) The name of the result should be the same as the name of the FUNCTION. The name of any result will not be same as the name of the SUB-ROUTINE. (3) The FUNCTION name should be in the same mode as the significant varibles in the DUMMY ARGUMENTS. This is not necessary in the case of SUB-ROUTINE'S name. (4) The FUNCTION is Invoked in the main program by its name whereas the SUB-ROUTINE is invoked by a CALL INSTRUCTION, eg,

```
CALL TRNSPS (B,I,J)
```

The program under Example 14.6 illustrates the use of a SUB-ROUTINE SUB-PROGRAM.

EXAMPLE 14.6 Write a Sub-routine Sub-program to check a vector of integer numbers of size N to find how many elements are equal to or above a value K and also to store such values in another array. Use this Sub-routine in the program for the following job.

There are 2 cards each of following format

COLUMN 1–4	Identification of Batsman (A4)
5	Blank
6–80	Runs scored in 25 innings, each score in 3 columns (I_3)

Write a program to print the number of centuries scored by each batsman and also to print the list of scores of 100 or above made of each. Also, compare their performance with respect to the centuries scored.

```
C        USE OF SUB-ROUTINE SUB-PROGRAM
C        SUB-ROUTINE TO SEARCH A VECTOR
         SUB-ROUTINE SRCH(INTR, N, K, IHIGH, L)
         DIMENSION INTR(100)
         COMMON INTR, IHIGH, L
         L=0
         DO 5 I=1, N
         IF (INTR(I), LT. K) GOTO 5
C        ELSE
         L=L+1
         IHIGH (L)=INTR (I)
      5  CONTINUE
         RETURN
         END
```

```
C         MAIN PROGRAM FOLLOWS
          INTEGER RUN1, RUN2, CNTRY1, CNTRY2
          DIMENSION RUN1 (25), RUN2 (25)
          CNTRY 1 (25), CNTRY2 (25)
          COMMON RUN1, CNTRY1, NCNTRY1
          EQUIVALENCE (RUN1, RUN2), (CNTRY1, CNTRY2)
          READ (5,10) IDN1, (RUN1(I), I=1, 25)
     10   FORMAT (A4, 1 x, 25I3)
          CALL SRCH (RUN1, 25, 100, CNTRY1, K)
C         THE ABOVE STATEMENT WILL INVOKE
C         THE SUB-ROUTINE SEARCH WHICH WILL
C         CHECK FOR RUNS ≥100 IN THE 25
C         ELEMENTS OF RUN1
C         IF NUMBER OF CENTURIES SCORED IS NOT ZERO
C         PRINT SCORE 100 AND ABOVE
          IF (K. EQ.0) GOTO 20
          ELSE
          WRITE (6,15), IDNI, (CNTRY1(J), J=1,K)
     15   FORMAT (6K, A4, 10 (5X, I3))
          GOTO 30
     20   WRITE (6,25), IDNI
     25   FORMAT (6X, A4, 'b̸ HAS SCORED NIL CENTURIES')
     30   READ (5,10), IDN2,,(RUN2(I),I=1,25)
          CALL SRCH (RUN2, 25, 100, CNTRY2, L)
          IF (L, EQ.0) GOTO 40
C         ELSE
          WRITE (6,15), IDN2, (CNTRY2 (J), J=1,L)
          GOTO 40
     35   WRITE (6,25) IDN2
     40   IF (K, EQ.0), AND IF (L.EQ.0) GOTO 75
C         ELSE
          IF (K-L) 45, 65, 55
     45   WRITE (6,50) IDN2, L
     50   FORMAT (6X, A4, 'b̸ HAS SCORED MORE CENTURIES,
          NAMELY b̸', I2)
          GOTO 75
     55   WRITE (6,50), IDN1, K
          GOTO 75
     65   WRITE (6,70), IDN1, IDN2, K
     70   FORMAT (6X, 'BOTH b̸', A4, 'b̸ AND b̸', A4,
          'HAVE SCORED EQUAL NUMBER OF CENTURIES, NAMELY b̸',
          I2)
     75   STOP
          END
```

Note that, in the above program, provision is made to output the values of INDEX VARIABLES K and L.

A SUBROUTINE can be used for a large variety of jobs, such as matrix operations, sorting of vectors, polynomial operations etc. Also, it can call upon another sub-program (FUNCTION or SUB-ROUTINE) or an Arithmetic statement function and built-in-functions.

14.2 SPECIFICATION STATEMENTS

SPECIFICATION statements are under the category of non-executable statements. Such statements are (i) FORMAT statements (ii) DIMENSION statements (iii) MEMORY MANAGEMENT statements (iv) TYPE statements (v) DATA statement, and (vi) EXTERNAL statements.

14.2.1 FORMAT Statement

The FORMAT statement specifies the lay-out of input record or an output record. It is normally a non-executable statement. But in case of Print-outs, the first character in the FORMAT is used as a CARRIAGE CONTROL instruction (refer 10.4.2)

14.2.2 DIMENSION Statement

DIMENSION statement has to be made in case of reading, processing and writing subscripted variables (vectors and arrays). Since all the elements in an array or vector are denoted by a single name but with different subscripts, the computer has to be intimated to allocate sufficient number of memory locations for storing all the elements in the array. This is done by the use of DIMENSION statements (refer Section 9.4.3.2)

14.2.3 MEMORY MANAGEMENT/ALLOCATION CONTROL Statements

The COMMON and EQIVALENCE statements fall under this category. They have been already discussed in detail under Section 14.1.6.

14.2.4 TYPE Statements

These statements explicitly specify the type of variable. There are six TYPE statements, namely (1) INTEGER, (2) REAL, (3) DOUBLE PRECISION, (4) LOGICAL and (5) COMPLEX.

INTEGER indicates that the variables listed are fixed point (INTEGER) variables (refer Section 9.4.2.2)

REAL indicates that the variables listed are FLOATING POINT (REAL) variables (refer Section 9.4.2.2.)

The above 2 definisions override the alphabetic naming convention.

But if a variable or a function is declared explicity of a given type, it will remain of that type throughout the program. Its type cannot be changed in the middle of the program by the use of IFIX or FLOAT function.

DOUBLE PRECISION indicates that the REAL variable listed are double-precision (16 digit accuracy) variables (refer Section 9.4.2.2.)

LOGICAL indicates that the variables are logical variables and, as such, can assume only 2 values, namely, TRUE or FALSE. (refer 12.4.4)

14.2.4.1 COMPLEX Variables

COMPLEX indicates that the variables indicated are complex variables. Only real numbers can be used in complex mode of operation.

eg,

COMPLEX V,C,P.

For processing complex numbers, **FORTRAN IV provides the LIBRARY FUNCTIONS CABS, REAL,** and AIMAG. (refer Section 11.3.2). The result of any Arithmetic operation on complex variables will also be a complex number.

EXAMPLE 14.7 In the A.C. Circuit given below Z_1 and Z_2 are impedances whose values in complex form are recorded on card in the format 2 F7.2 and 2 F7.3 respectively. They are connected across 230V. Compute the effective value of the impedance, the current, power, power factor and power factor angle, as also the voltages V_1 and V_2 across Z_1 and Z_2, respectively, and print them.

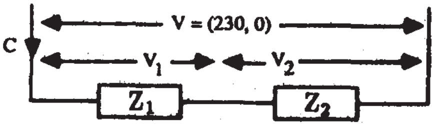


Fig 14.4

Solution:

Effective impedance=$Z = Z_1 + Z_2$

Current C=V/Z(VECTORIALLY)

$$\text{PF} = \text{COS(PHI)} = \frac{\text{REAL(C)}}{\text{CABS(C)}}$$

POWER=P=CABS(V)*CABS(I)*PF

$V_1 = \text{CABS}(C * Z_1)$

$V_2 = \text{CABS}(C * Z_2)$

PHI=PF ANGLE=ATAN (AIMAG(C)/REAL(C)) IN RADIANS

Program

```
C              PROCESSING COMPLEX CONSTANTS AND
C              VARIABLES
               COMPLEX V, C, Z1,Z2,Z
               READ (5,10) Z1,Z2
    10         FORMAT (2 F7.2, 2 F7.3)
               Z = Z1 + Z2
               V=(230, 0)
               C=V/Z
               CEFF=CABS (C)
               ZEFF=CABS (Z)
               PF=REAL(C)/CEFF
               P = 230* CEFF*PF
               V1=CABS (C * Z1)
               V2=CABS (C * Z2)
               PHI=ATAN (AIMAG(C)/REAL (C))*57.29578
          0    WRITE (6,15), ZEFF, CEFF, PF, PHI,
          1    P,V1,V2
```

```
15 | 0 | FORMAT (6X, 'EFFECTIVE IMPEDANCE=',
   | 1 | F8.3, 'b̸ OHMS', 5X, 'EFFECTIVE b̸
   | 2 | CURRENT=', F7.3, 'b̸ AMPS'/
   | 3 | 6X, 'POWER FACTOR=', F7.5, 5X,
   | 4 | 'POWER=', F9.2, 'b̸ WATTS',/
   | 5 | 6X, 'V1=', F7.3, 'b̸ VOLTS', 5x
   | 6 | 'V2=', F7.3, 'b̸ VOLTS')
   |   | STOP
   |   | END
```

14.2.5 Data Statements

DATA statements are used to define variables in the source program. This provides for data to be placed in the object program during compilation. If the data variables are redefined during program execution, they will assume their new values regardless of the value given in the DATA statement. A typical DATA statement may be

DATA I, A, DoT, LINE/3, 4.2, '.', 100*'–'/

The above DATA statement means:

Initialise I to value 3,

Initialise A to value 4.2.

The word DoT in programs is the character '.' and the word LINE means 100 successive characters each '-'.

The DATA statements are useful while drawing graph using the ordinary printer.

Since the carriage movement of the printer is top to bottom vertically and left to right horizontally (and cannot be in reverse to above directions), the vertical axis is labelled as X axis and the horizontal axis as Y axis. Any increment in X should correspond to a line space and the corresponding value of Y should be truncated to an integer and expressed in terms of the character position on the horizontal line. Suitable scales on X and Y axes may have to be used to space the graph properly within a printing page. The following example is illustrative of these.

EXAMPLE 14.8 WRITE A PROGRAM to plot the curve $y = x^3 - 2x^2 + 3x - 4$ for x ranging from -3 to 3 using an 80 character printer.

Solution:

Since a page contains 72 lines -3 to $+3$ can be accommodated in 61 lines, the middle one eg, 31st line will be the Y axis. The increment in x will be 0.1 which will correspond 1 line space

(ie,) $\frac{\text{UPPER LIMIT}-\text{LOWER LIMIT}}{\text{NO OF LINES}-1} = \frac{(+3-(-3))}{60} = 0.1$

By calculation, the minimum value of y will be -55 and maximum value 14. These can be accommodated in 71 character positions, the 51st position will be X axes. The 2 axes will be drawn by dots and the graph by asterisk marks (refer Fig. 14.5).

Solution:

```
C           PLOTTING A CURVE
            INTEGER DOT, STAR YAXIS, CURVE,
          0 DATA BLANK, DOT, STAR, YAXIS, CURVE/
          1 ‘b̸’, ‘.’, ‘*’, 71*‘.’ ‘71 *‘h’/
            DIMENSION YAXIS (71), CURVE (71)
C           START GIVING VALUES TO X
            X=-3.0
C           COMPUTE VALUE OF Y
          3 Y=((x-2.0)*x+3.0)*x-4.0
C           GET CORRESPONDING CHARACTER POSITION
            M=INT (y+56.5)
C           56 IS ADDED TO GET THE CHARACTER POSITION
C           1 TO 80 AND 0.5 FOR ROUNDING OF
C           CHECK IF IT IS Y AXIS IF SO PRINT IT
C           WITH DOTS
C           IF (ABS(X), LT. 0.0001) GOTO 10
C           ELSE
            CURVE (51)=DOT
C           DOT REPRESENTS X AXIS
            CURVE (M)=STAR
            WRITE (6,5) CURVE
     5      FORMAT (1X, 71A)
            CURVE (M)=BLANK
C           THE ABOVE STEP IS TO PREVENT THE
C           STAR FROM BEING CARRIED TO NEXT LINE
            GOTO 20
    10      DO 15 I=1, 71
    15      Y AXIS (I)=DOT
            Y AXIS (M)=STAR
            WRITE (6,5); Y AXIS
    20      X=X+0.1
            IF (X, LE. 3.0) GOTO 3
C           ELSE
            STOP
            END
```

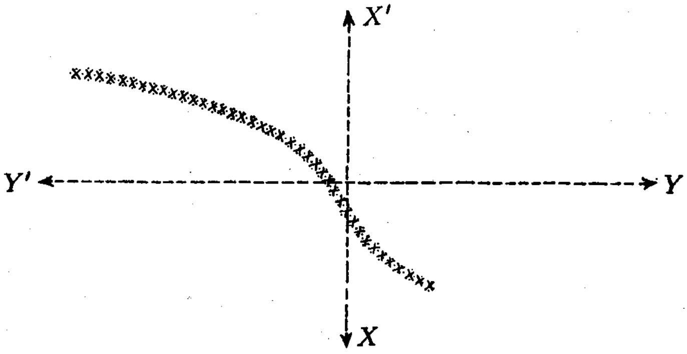


Fig 14.5

14.2.6 External Statement

EXTERNAL used in a program means that the names listed after EXTERNAL are defined outside that program. This facilitates sub-program names to be used as arguments in a sub-routine or function.

eg,

```
EXTERNAL NFACL, SRCH
```

14.3 ORDER OF STATEMENTS IN A PROGRAM

The program statements will normally be in the following order:

(1) TYPE statements if any;

(2) DIMENSION statements if any;

(3) COMMON statements if any;

(4) EQUIVALENCE statements if any;

(5) INPUT statements (EXECUTABLE);

(6) FORMAT statement For the input;

(7) PROCESSING statements (EXECUTABLE);

(8) OUTPUT Statement (EXECUTABLE);

(9) FORMAT Statement for the output;

(10) STOP Statement (EXECUTABLE) (or RETURN Statement if Sub-Program);

(11) END Statement.

Summary

When certain operations are to be repeatedly performed and BUILT-IN-FUNCTIONS for performing them are not available, user can prepare SUB-PROGRAMS.

FORTRAN allows for 3 types of SUB-PROGRAMS, namely (1) ARITHMETIC STATEMENT Function, (2) FUNCTION SUB-PROGRAM and (3) SUB-ROUTINE SUB-PROGRAM.

When a single statement giving a single result is to be repeatedly used, ARITHMETIC STATEMENT function is used. If a group of steps giving a single result is to be repeatedly used, a FUNCTION SUB-PROGRAM is used.

If a group of steps giving one or more results is to be used repeatedly, SUB-ROUTINE SUB-PROGRAM is used.

SPECIFICATION statements are non-executable statements. The TYPE statements, DIMENSION statements, COMMON statements, EQUIVALENCE statements and FORMAT statements are all of this category. Other specification statements are DATA and EXTERNAL statements.

Exercise/Questions

14.1 Write an Arithmetic statement function for the statement

$$D = \sqrt{2x^3 + 4y^2} + 1.63e^{(x+y)}$$

14.2 What are (a) closed sub-programs and open sub-programs? (b) Value parameter, and variable parameters?

14.3 Rewrite the program under Exercise 14.5, using a function sub-routine to find area by Simpson's rule.

14.4 Write a program to plot a sine curve for $\theta = -180°$ to $180°$ using a Line-Printer.

14.5 Write a Sub-routine Sub-program to compute mean, median and standard deviation of a given array.

14.6 Rewrite the program under Example 13.12 using a Sub-routine Sub-program to read a matrix.

14.7 Rewrite the programs under Examples 14.4, 14.5 and 14.6 using a Sub-routine Sub-program to read the vectors.

for NOTES

Chapter Fifteen

Additional Programming Exercises

Exercise/Questions

15.1 A body is projected with an initial velocity V_0 at horizontal angle θ degree, from a point of height H meters.

Assume that the values of V_0, θ and H are recorded on card with suitable format. Write a program to find the instantaneous height and velocity of body at the interval 0.02 seconds until it reaches the ground level. The total horizontal distance covered should also be computed and printed.

15.2 Given 3 sides, it is to be checked whether they can constitute a triangle. If so, find the angles and also the type of the triangle.

15.3 There are 100 records each giving the REGNO ($I5$) of a student and the marks ($I3$) secured by him. Write a program to print the table of Reg no. and marks in the ascending order of Reg no.

15.4 Write a program to print the table in the previous problem in the descending order of marks.

15.5 Write a program to find all the 3 roots of the cubic equation:

$$x^3 - 4x^2 + 5x - 2 = 0$$

[**HINT.** Find one root by Newton Raphson method. Using it, reduce the given equation to a quadratic equation and solve it to get the other 2 roots.]

15.6 Write a program to evaluate a determinant of size 4 × 4 by method of triangulation.

15.7 Write a program to obtain the transpose of a given matrix of size $m \times n$.

15.8 Given an array called A containing 50 numbers, write a program to compute the polynomial: $4Ai^4 - 5Ai^3 - 3Ai^2 + 14$ for all values of the A array. If the computed value of the polynomial is positive, place it in an array called APOS and otherwise in an array called ANEC.

15.9 Given a vector of 40 real number, write a program to compute their STANDARD DEVIATION and variance.

15.10 According to STIRLING approximation for Factorial, Factorial of a number n is given approximately as:

$$n! = e^{-n}(n)^n\sqrt{2\pi n}$$

Write a program to read a single digit number and to obtain the factorial of it by the conventional method as well as by the STARLING approximation and to estimate the error in the STARLING method.

15.11 Write a program to read a 2 digit integer number and compute its factorial using the method:

$$\log(N!) = \log(1) + \log(2) + \cdots + \log(N).$$

15.12 Write a program to get the inverse of a given matrix of size $M \times M$.

15.13 A table contains 25 sets of values of x and y. Write a program to compute their CORRELATION COEFFICIENT.

15.14 Write a program to read the value of θ in degrees and compute $\cos\theta$ and $\sin\theta$ by series expansion.

15.15 Write a program to read the value of an integer number x and evaluate e^x by series expansion.

15.16 Assuming that $N > R$ and both of them are integers, write a program to compute ${}^{N}C_R$. Use a FUNCTION subprogram to compute factorial.

15.17 Write a program to print both the maximum and minimum elements in an unsorted vector of REAL numbers.

15.18 Write a program to read the values of a and x both of format (F 7.5) and n of format (I_1) and compute the binomial expansion $(a + x)^n$.

15.19 Given the values of E, R, L, C and F (all real variables), write a program to compute

$$I = \frac{E}{\sqrt{R^2 + \left(2\pi FL - \frac{1}{2\pi FC}\right)^2}}$$

I is also a real variable.

15.20 Given a group of simultaneous equations with 3 variables x_1, x_2 and x_3, write a program to solve them by the use of CRAMER's rule.

15.21 There are 50 values of θ in degrees with 10 values in each card, each in the form of F 8.4. Write a program to compute $A = \tan^{-1}(\cos^2\theta + \tan^3\theta)$ for each value of θ and print the value of θ and A in each case.

[STUDENTS are advised to write programs also for the numerical methods discussed in the next section.]

for NOTES

"This page is Intentionally Left Blank"

Part C

Introductory Numerical Analysis

Chapter Sixteen Approximation, Significant Digits and Error *171 – 177*
Chapter Seventeen Linear, Algebraic and Transcendental Equations *178 – 195*
Chapter Eighteen Simultaneous Algebraic Equations (Matrix Inversion) *196 – 212*
Chapter Nineteen Eigen Values and Eigen Vectors *213 – 222*
Chapter Twenty Numerical Integration *223 – 229*
Chapter Twentyone Numerical Solution to Ordinary Differential Equations *230 – 241*

"This page is Intentionally Left Blank"

Chapter Sixteen

Approximation, Significant Digits and Error

16.1 COMPUTER ORIENTED NUMERICAL METHODS

As explained in Chapter 8, the method of solution of problems by the computer is often different from that adopted in the Mathematics classes. This is because the digital computer cannot perform, as such, differentiation or integration or evaluate trigonometric and logarithmic functions. So, numerical methods have been evolved to get the algebraic equations, simultaneous equations, differential equations etc solved by the digital computer. Such numerical methods are called COMPUTER ORIENTED NUMERICAL METHODS.

16.2 APPROXIMATION

Calculations involving numerical methods are subject to approximation due to following reasons: (1) constraint imposed by the processor on the accuracy of representation of numbers, eg, in FORTRAN, the accuracy will be of 7 digits only, in single precision operation and 16 digits in double precision operations. (2) Constraint imposed by the memory: The memory can store only fixed word-lengths, say 16 or 32 bits of binary data.

16.3 SIGNIFICANT DIGITS

When a number is approximated, it is expressed in specific number of SIGNIFICANT DIGITS.

The digits that are used to express a number are called SIGNIFICANT DIGITS or SIGNIFICANT FIGURES.

eg, Nos. 3.412854, 35.82561, 0.7257421, 0.005623414 are all numbers of 7 significant digits.

In single precision mode of processing in FORTRAN, if a number with 8 or more digits is come across, it will be approximated to 7 digits.

The process of approximation involves ROUNDING-OFF. The process of rounding-off can be stated as follows:

(i) If the number is to be rounded-off to n significant digits, discard all digits to the right of the nth digit, (ii) If the $(n+1)$th digit is less than half the value of the base or radix, leave the nth digit unaltered, (iii) Otherwise increase the nth digit by 1.

eg, 28.47238623 will be rounded to 28.47239 and 49.42617415 to 49.42617 if each is to be rounded-off to 7 digits.

16.4 ERRORS

Calculations involving numerical methods may cause three types of errors. They are (i) INHERENT ERROR, (ii) TRUNCATION ERROR, and (iii) ROUND-OFF ERROR.

An INHERENT ERROR may be due to (i) limitations in the process of collecting data, and (ii) limitation in the numerical method used. For example, while measuring the ordinates in a plot in order to find area, measurement can be made to the accuracy of millimeter only. So the values collected may be of less than 7 digits up to which the computer can process. This is an example of limitation in the process of collection of data.

If Simpson's One-Third Rule is used to compute the area of the above plot, it is based on the assumption that the curve enclosing two adjacent segments is parabolic which may not be true in many cases. This is an example of limitation in the numerical method.

16.4.1 Truncation Error

Truncation means limiting the result to the allowed number of digits without rounding, eg, 27.8294578 will be 27.82945 if truncated to 7 digits. Truncation error will occur in computations either when (i) an infinite mathematical process is approximated by a finite process, or (ii) when a finite process is approximated by a smaller number of iterations.

If, for example, calculations of $\cos x$(x in radians) using TAYLOR series,

$$\cos x = 1 - x^2/2! + x^4/4! - x^6/6! + \cdots\cdots \text{ etc}$$

there are 2 methods of ending the summation of the terms in such series. One method is to stop the summations after definite number of iterations, ie, truncating the formula after definite number of terms. Then, the calculated value will be inexact. This is an example of truncation error due to approximating an infinite process into a finite process.

The other method of stopping the summation of an infinite series is when the terms are in the decreasing order and when the absolute value of the term becomes less than a certain value, say, 0.0000001. In this case also, the resulting sum will not be exact. This is an example of truncation error due to approximating an infinite process into a smaller number of iterations.

16.4.2 Round-Off Errors

As discussed in Section 16.3, rounding-off is to be done to express a number in specific number of significant digits. Since the word-size of a computer is fixed, rounding-off is essential and hence the occurrence of ROUND-OFF errors frequently.

16.5 MEASURE OF ERRORS

Since errors are bound to occur in most of the computations, there is a need to measure the accuracy of results. This is done by measuring the magnitude of errors.

An error can be measured as (i) ABSOLUTE ERROR, or (ii) RELATIVE ERROR, or (iii) PERCENTAGE ERROR.

Let x be the exact value of a result and x_a be its computed approximate value.

$$\text{ABSOLUTE ERROR} = |x - x_a| = e_a$$

$$\text{RELATIVE ERROR} = \left|\frac{x - x_a}{x}\right| \approx \left|\frac{x - x_a}{x_a}\right| = e_r = \frac{e_a}{x_a}$$

$$\text{PERCENTAGE ERROR} = \left|\frac{x - x_a}{x_a}\right| \times 100 = e_p = e_r \times 100$$

EXAMPLE 16.1 IF 34.127934 is rounded to 34.12793, estimate the absolute error, relative error and percentage error.

Solution:

$$e_a = x - x_a = 34.127934 - 34.12793 = 0.000004$$

$$e_r = \frac{e_a}{x_a} = \frac{0.000004}{34.12793} = 0.00000011$$

$$e_p = e_r \times 100 = 0.00000011 \times 100 = 0.000011\%.$$

A general formula to pre-estimate rounding-off errors is given as :

$e_a = \frac{1}{2}10^{-n}$ where the number is to be rounded-off to n decimal places. In the above case if this formula is applied:

$$e_a = \frac{1}{2}10^{-5} = 0.000005$$

From this e_r and e_p can be evaluated.

16.6 PROPAGATION OF ERRORS

When computing a result involves a number of Arithmetic operations, each operation having its result rounded-off, these errors will have a cumulative effect on the final result. This is called ACCUMULATION or PROPAGATION of ERRORS.

16.6.1 Error Propagation in Arithmetic Operations

(a) ADDITION

Assume that an addition is to be performed on the acutual values denoted as x and y and their computed values are x_a and y_a. The result will be $(x_a + y_a)$ instead of $x + y$.

e_x = absolute error in $x = x - x_a$

e_y = absolute error in $y = y - y_a$

e_a = absolute error in the result $= (x + y) - (x_a + y_a) = (x - x_a) + (y - y_a) = e_x + e_y$

e_r = relative error in the result $= \dfrac{e_x + e_y}{x_a + y_a}$

(b) SUBTRACTION

Assuming same variables x, y and x_a, y_a and $(x - y)$ is to be computed, the result will be $(x_a - y_a)$ instead of $(x - y)$

$$e_a = \text{absolute error in the result} = (x - y) - (x_a - y_a) = e_x - e_y$$
$$e_r = \text{relative error} = \frac{e_x - e_y}{x_a - y_a}$$

(c) MULTIPLICATION

In this case, the result will be $(x_a.y_a)$ instead $(x.y)$

$$\begin{aligned} x.y &= (x_a + e_x) \cdot (y_a + e_y) = x_a.y_a + e_x.y_a + e_y.x_a + e_x.e_y \\ e_a &= (x.y) - (x_a.y_a) = \text{ABSOLUTE ERROR} \\ &= e_x.y_a + e_y.x_a \text{ (neglecting } e_x.e_y \text{ which will be relatively small)} \\ e_r &= \frac{e_x.y_a + e_y.x_a}{x_a.y_a} = \text{RELATIVE ERROR} \\ &= \frac{e_x}{x_a} + \frac{e_y}{y_a} = \text{SUM of relative errors in } x \text{ and } y \end{aligned}$$

(d) DIVISION

In this case, the result obtained will be x_a/y_a instead of x/y.

$$x/y = \frac{x_a + e_x}{y_a + e_y} = \frac{x_a + e_x}{y_a + e_y} \times \frac{y_a - e_y}{y_a - e_y} = \frac{x_a y_a + y_a.e_x - x_a.e_y - e_x.e_y}{y_a^2 - ey^2}$$

Neglecting $ex \cdot ey$ and e_y^2 since they will be too small.

$$\frac{x}{y} = \frac{x_a y_a + y_a e_x - x_a e_y}{y_a^2} = \frac{x_a}{y_a} + \frac{e_x}{y_a} - \frac{x_a e_y}{y_a^2} = \frac{x_a}{y_a} + \frac{x_a}{y_a}\left(\frac{e_x}{x_a} - \frac{e_y}{y_a}\right)$$
$$\therefore \quad e_a = \frac{x_a}{y_a}\left(\frac{e_x}{x_a} - \frac{e_y}{y_a}\right) = \text{ABSOLUTE ERROR}$$
$$e_r = e_a \div x_a/y_a = \frac{e_x}{x_a} - \frac{e_y}{y_a} = \text{RELATIVE ERROR}$$

This is the difference between the errors of numerator and denominator.

16.6.2 A General Error Formula

If a result is to be computed using 3 or more variables and besides Arithmetic operations, functional relations such as (SIN, COS, LOG etc) are involved in the computation, the absolute error can be found by the use of a general formula as below—

Let $W = f(x_1, x_2, x_3 \cdots x_n)$ be computed

Let W_a = be the result of computation

Let e_a = absolute error = $W - W_a$

e_a can be written ΔW

Let x_1a, x_2a etc be approximately computed values of x_1, x_2 etc respectively and e_1a, e_2a etc be, respectively, the correspponding errors in each case. Let e_1a, e_2a, e_3a etc be represented as $\Delta x_1, \Delta x_2, \Delta x_3$ etc

$$W = W_a + \Delta W_a = f(x_1, x_2, x_3 \cdots x_n) = f(x_{1a} + \Delta x_1, x_{2a} + \Delta x_2 + \cdots, x_{na} + \Delta x_n)$$

Expanding by Taylor's series,

$$\begin{aligned} W + \Delta W_a = W &= f(x_{1a}, x - 2a, \cdots x_{na}) \\ &+ \sum_{c=1}^{n} \frac{\partial f}{\partial x_i} \Delta x_{ia} + \text{(other terms involving powers of } \Delta x_{ia} \text{ which can be neglected)} \\ \Delta w_a &= \sum_{c=1}^{n} \frac{\delta f}{\delta ei} \Delta x i_a = e_a = \text{ absolute error} \end{aligned}$$

relative error $= e_r = \dfrac{e_a}{W_a}$.

EXAMPLE 16.2 Assuming that the values of x, y and z lie near about 2, 4 and 3, respectively, and their individual absolute errors are .001, .002 and .0015, respectively. Find the maximum possible error in the computation of

$$W = \frac{4x^2y}{z^2}$$

Solution:

$$\frac{\partial w}{\partial x} = \frac{8xy}{z^2}; \quad \frac{\partial w}{\partial y} = \frac{4x^2}{z^2}; \quad \frac{\partial w}{\partial z} = \frac{-8x^2y}{z^3}.$$

$$\begin{aligned} \text{ABSOLUTE ERROR IN RESULT} = e_a &= \frac{\partial f}{\partial x} \cdot \Delta x + \frac{\partial f}{\partial y} \cdot \Delta y + \frac{\partial f}{\partial z} \cdot \Delta z \\ &= \frac{8xy}{z^2} \cdot \Delta x + \frac{4x^2}{z^2} \cdot \Delta y - \frac{8x^2y}{z^3} \cdot \Delta z \end{aligned}$$

Since the maximum possible error will be dependent on absolute values only,

$$e_a = \frac{8xy}{z^2} \cdot \Delta x + \frac{4x^2}{z^2} \cdot \Delta y + \frac{8x^2y}{z^3} \cdot \Delta z = \frac{8 \times 2 \times 4}{3^2} \times .001 + \frac{4 \times 2^2}{3^2} \times .002 + \frac{8 \times 2^2 \times 4}{3^3} \times .0015$$

$$= 0.0071 + 0.0036 + 0.0078 = 0.0185$$

$$W = \frac{4x^2y}{z^2} = \frac{4 \times 2^2 \times 4}{6} = \frac{64}{6} = 10.67$$

$$e_r = \frac{e_a}{W} = \frac{0.0185}{10.67} = 0.00173.$$

Maximum value of error = 0.0185

Relative error = 0.00173.

16.7 SERIES APPROXIMATION

In Section 16.4.1, it was discussed that truncation has to be done at a particular stage while finding the Sum of any infinite series. The particular stage is indicated by the number of terms which are to be summed. This number of terms depends on the desired accuracy in terms of the number of digits. This can be evaluated by using the remainder after 'n' terms, where n is the number of terms up to and inclusive of which summation will be done to get the accuracy of desired number of digits.

Using Taylor's series for $f(x)$ at $x \approx a$, which will be:

$$f(x) = f(x) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''a + \text{etc} \cdots + \frac{(x-a)^{n-1}}{(n-1)!}f^{n-1}(a) + R_n(x)$$

where $R_n(x)_n$ is called remainder term.

In case if $f(x)$ is to be approximated up to and inclusive of nth term the error will be the remainder term $Rn(x)$, where

$$R_n(x) = \frac{(x-a)^n}{n!}f^n\xi$$

where $a < \xi < x$

EXAMPLE 16.3 Estimate the number of terms to be summed up to evaluated e^x up to the accuracy of 7 digits. Assume $x \approx 1$.

Solution:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^{n-1}}{(n-1)!} + Rn(x)$$

where $Rn(x) = \frac{x^n}{n!}e\xi$ where $0 < \xi < x$

$Rn(x)$ = error term.

This is maximum where $\xi = x$ in this case

So error term = absolute error = $\frac{x^n}{n!}e^x = \frac{e}{n!}$. (Since $x \approx 1$)

Since 7 digit accuracy is required, the error term should be less than or equal to $0.00000005 = \frac{1}{2} \times 10^{-7}$.

Equating $\frac{e}{n!} = \frac{1}{2} \times 10^{-7} = \frac{2.7183}{n!} \leq \frac{1}{2} \times 10^{-7}$; $n = 10$

ie, The series is to summed up to 10 terms to get the sum accurate of 7 digits.

Summary

For solving numerical problems using the computer, special methods have to be formulated. Such methods involve approximation which will result in errors in computations. These errors are—inherent errors, truncation errors and round-off errors. The error is measured by computing absolute error or relative error or percentage error. The probable errors in computation can be predetermined and can be used to correct the results obtained from the computer.

Exercise/Questions

16.1 TRUNCATE the following numbers up to 7 significant digits.

(1) 38.53278953 (2) −42.42796923 (3) −54.29738267 (4) 35.69425149.

16.2 ROUND-OFF the above numbers to 7 significant digits.

16.3 Explain why special numerical methods are required for solving problems with the use of digital computers.

16.4 What are the types of errors in the computer-oriented numerical methods? What is the cause for errors?

16.5 Explain how the errors in computation can be measured.

16.6 Explain the term 'PROPAGATION OF ERRORS.'

16.7 Using TAYLOR's Series formula for $\sin x$, find out up to how many terms the summation is to be done to get the result to the accuracy of 7 digits when $x \approx 1$ radian.

for NOTES

Chapter Seventeen

Linear, Algebraic and Transcendental Equations

17.1 INTRODUCTION

In this chapter, some of the numerical methods to obtain the roots of (a) LINEAR EQUATIONS, (b) ALGEBRAIC EQUATIONS, and (c) TRANSCENDENTAL EQUATIONS are discussed.

17.2 LINEAR EQUATIONS

A LINEAR FUNCTION is of the form:

$f(x) = mx + c$ (refer Fig 17.1)

If this function is equated to ZERO, it will be a LINEAR EQUATION.

$mx + c = 0$ is a linear equation. The solution to a linear equation can be obtained very easily as follows

$$mx = -c; \quad x = -c/m$$

solution to a linear equation is simple and requires no special methods.

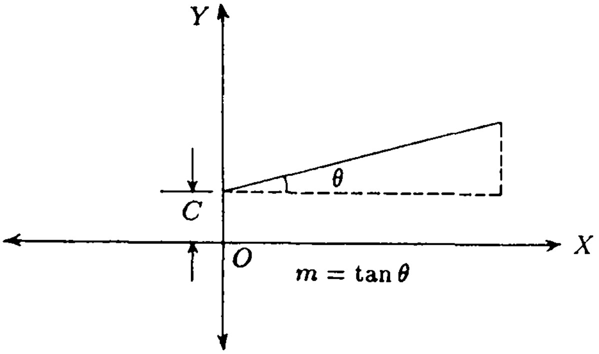


Fig 17.1

17.3 ALGEBRAIC EQUATIONS

An algebraic function is of the form

$$f(x) = x^n + C_1 x^{n-1} + C_2 x^{n-2} + \ldots + C_{n-1} x + C_n$$

$$\text{eg,} \quad y = f(x) = x^4 - 5x^3 + 4x^2 - 3x + 6$$

These are also called POLYNOMIAL functions with single variable, say, x. When an algabraic function is equated to zero, it becomes an ALGEBRAIC EQUATION. eg,

$x^4 - 5x^3 + 4x^2 - 3x + 6 = 0$ is a polynomial or Algebraic equation.

17.4 TRANSCENDENTAL EQUATIONS

A transcendental function is of the form

$$f(x) = \sum_{n=0}^{\alpha} c_i x^i, \quad \text{eg,} \quad y = f(x) = x - \tan x$$

$$\text{or} \quad y = f(x) = e^x + \sin x \text{ etc.}$$

Note that each of such transcendental functions has a 'series expansion form'.

A transcendental function equated to zero is a transcendental equation. eg, $e^x + \sin x = 0$ is a transcendental equation.

Both the algebraic and transcendental equations contain only one variable (say x)

17.5 NUMERICAL METHODS

For solving algebraic and transcendental equations with the help of computers, the following numerical methods are useful—

(i) The incremental search method;

(ii) The method of bisection;

(iii) Regula falsi method;

(iv) Secant method; and

(v) Newton-Raphson method.

These methods are all iterative. An ITERATIVE METHOD consists of repeated execution of the same procedure in which, at each stage, the result of the previous stage is used to compute more accurate values of the result.

17.5.1 Incremental Search Method

This is also called TRIAL AND ERROR method. This can be used to find a single (positive or negative) root of an equation.

In this method, x is given different values and at each value of x, the corresponding value of $f(x)$ is calculated. The two successive values of x, say, x_1, and x_2 between which $f(x)$ changes, sign are noted. (The change in sign indicates that a root has been passed). Then the value of x, say, x_1

preceding the change in sign is taken. Beginning with this value and using smaller increment, (say $\frac{1}{10}$th of the previous increment) the values of $f(x)$ are determined until there is a change in sign again.

The above is repeated with smaller increments each time, until the root to the desired accuracy is found.

EXAMPLE 17.1 Determine the first positive non-zero root of the following equation

$$x^2 - 0.75x - 3.375 = 0.$$

Solution:

ASSUME $x = 0$

$$f(x) = 0 - 0.75 \times 0 - 3.375 = -3.375$$

Increment x by 1, ie, $x = 1$

$$f(x) = 1^2 - 0.75 - 3.375 = -3.125$$

Now make $x = 2$

$$f(x) = 2^2 - 0.75 \times 2 - 3.375 = -0.875$$

Making $x = 3$

$$f(x) = 3^2 - 0.75 \times 3 - 3.375 = 3.375$$

There is a change in sign between $x = 2$ and $x = 3$.

So, one root of the equation lies between 2 and 3.

Now, make $x = 2$ again

$$f(x) = -0.875$$

Increment x by 0.1 and repeat the above procedure (the increment is $\frac{1}{10}$th of increment in the previous cycle of operations) making $x = 2.1$

$$f(x) = 2.1^2 - 0.75 \times 2.1 - 3.375 = -0.54.$$

Increment x again by 0.1

$$x = 2.2$$

$$f(x) = 2.2^2 - 0.75 \times 2.2 - 3.375 = -0.185$$

Now, making $x = 2.3$

$$f(x) = 2.3^3 - 0.75 \times 2.3 - 3.375 = 0.19.$$

Now there is a change in sign between the values $x = 2.2$ and $x = 2.3$. So one root lies between 2.2 and 2.3.

Now, make $x = 2.2$ again

$$f(x) = -0.185$$

Now, the incrementation is 0.01, ie, $\frac{1}{10}$th of previous and the process is repeated.

Making $x = 2.21$

$$f(x) = 2.21^2 - 0.75 \times 2.21 - 3.375 = -0.1484$$

Making $x = 2.22$

$$f(x) = 2.22^2 - 0.75 \times 2.22 - 3.375 = -0.1116$$

Making $x = 2.23$

$$f(x) = 2.23^3 - 0.75 \times 2.23 - 3.375 = -0.0746$$

Making $x = 2.24$

$$f(x) = 2.24^2 - 0.75 \times 2.24 - 3.375 = -0.0374.$$

Making $x = 2.25$

$$f(x) = 2.25^2 - 0.75 \times 2.25 - 3.375 = 0.$$

$\therefore$ 2.25 is the root of the equation (refer Fig 17.2)

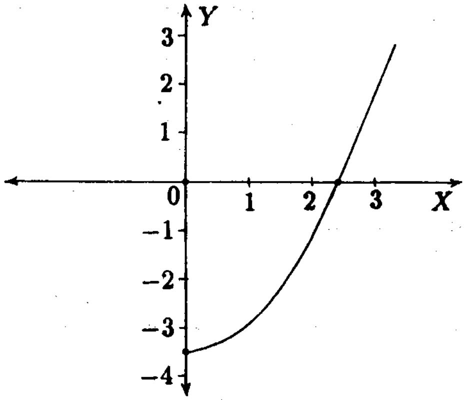


Fig 17.2

17.5.1.1 Algorithm Incremental Search

(Given x an initial guess of the root of $f(x)$ and the desired accuracy E as also an initial increment Dx in x, this algorithm is to find ROOT, a root of $f(x)$)

```
1. (INPUT VALUES OF X,DX and E)
   READ (X,DX,E)
2. (COMPUTE f(X))
   FX ← f(X)
3. (INCREMENT X)
   X ← X + DX
4. (COMPUTE $FX_1$ WITH NEW VALUE OF X)
   $FX_1$ ← f(X)
5. (Check if there is a change in sign)
   IF (FX * $FX_1$) > 0. Then GOTO 3
6. (ELSE)
   IF (FX * $FX_1$) < 0 Then
```

```
     IF (DX < E) Then GOTO 7
     ELSE (IF DX > E)
     DX ← $\frac{DX}{10}$
     FX ← $FX_1$
     GOTO 3
  7. ELSE (IF FX, $FX_1$ = 0)
     ROOT ← X
  8. (OUTPUT ROOT)
     WRITE ('ONE ROOT OF THE EQUATION', ROOT)
  9. EXIT.
```

17.5.2 The Method of Successive BISECTION

This method is also called the BOLZANO method or 'half-interval' method.

In this method, the two values of x, namely x_1 and x_2 between which $f(x)$ changes its sign, are found by, say, incremental search method. Further steps in this method are as follows—

(1) Bisect the interval x_1 and x_2 and denote the midpoint as x_m

$$x_m = (x_1 + x_2)/2$$

(2) Find $f(x_m)$. If it is of same sign as $f(x_1)$, then the root lies between x_m and x_2. Then make $x_1 = x_m$ and, with this value of x_1, find the new value of midpoint (x_m) and check the sign of $f(x_m)$ again.

(3) If $f(x_m)$ is of the same sign as $f(x_2)$, then the root lies between x_1 and x_m. In such case, make $x_2 = x_m$ and, with this value of x_2, find the new value of midpoint (x_m) and check again the sign of $f(x_m)$.

The above iterations are repeated until the exact value of the root is got or it is obtained to the required accuracy.

In this method, the interval in which the root is searched for is halved in each iteration and this gives the solution faster than the incremental search method.

EXAMPLE 17.2 SOLVE the example 17.1 by the method of successive bisections.

Solution:

The equation is

$$x^2 - 0.75x - 3.375 = 0$$

By incremental search method, it is found that one root lies between 2 and 3.

$$x_1 = 2 \text{ and } f(x_1) = -0.875$$
$$x_2 = 3 \text{ and } f(x_2) = 3.375$$

ROOT $= x_m = (x_1 + x_2)/2 = (2 + 3)/2 = 2.5$

$$f(x_m) = 2.5^2 - 0.75 \times 2.5 - 3.375 = 1.$$

This has the same sign as $f(x_2)$. So the root lies between x_1 and x_m, ie, between 2 and 2.5. Making $x_2 = x_m = 2.5$, new value of $x_m = (x_1 + x_2)/2 = (2 + 2.5)/2 = 2.25$.

Now $f(x_m) = 2.25^2 - 0.75 \times 2.25 - 3.375 = 0$.

2.25 is one of the roots of the given algebraic equation (ROOT = 2.25) (Refer Fig 17.3).

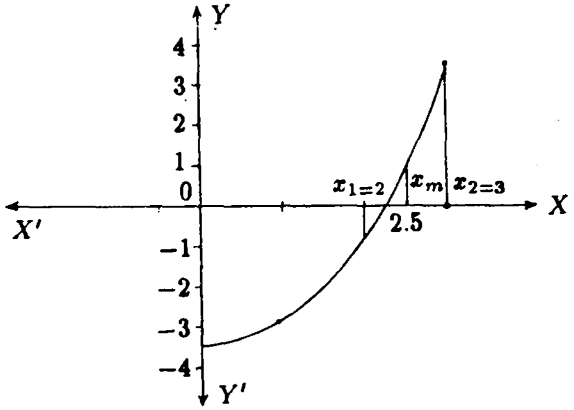


Fig 17.3

17.5.2.1 Algorithm : Successive Bisection

(Given two values of x, namely x_1 and x_2, such that $f(x)$ changes sign in-between them, and E the desired accuracy, to find ROOT which is the value of X that saisfies $f(x) = 0$)

```
1. (INPUT values of $X_1, X_2$ and E)
   READ ($X_1, X_2$, E)
2. (Check if $X_1 = X_2$)
   IF $|X_1 - X_2| < E$ Then GOTO 8
3. (Else find the midpoint of the interval)
   ELSE
   Root $\leftarrow (X_1 + X_2)/2$
4. $FX_1 \leftarrow f(X_1)$
   $FX_2 \leftarrow f(X_2)$
   FX $\leftarrow$ f(ROOT)
5. (check if ROOT is the desired one)
   IF (FX = 0) THEN GOTO 9
6. ELSE
   IF $(FX * FX_1) < 0$ THEN
   $X_2 \leftarrow$ ROOT
   GOTO 2
7. ELSE
   (IF $FX * FX_2 < 0$)
   $X_1 \leftarrow$ ROOT
   GOTO 2
8. ROOT $\leftarrow X_2$
9. (output ROOT)
   WRITE ('ROOT =', ROOT)
```

10. (Finished)
    EXIT

NOTE. In the above algorithm, to check whether there is a change in sign, the product of Fx and Fx_1 is checked as to whether it is negative. (Step 6). If both Fx and Fx_1 are positive or negative, this product will be positive. If it is negative, it indicates that they are of different signs, which, in turn, indicates a change in sign.

17.5.2.2 Corollary

The method of bisection can be used to find the nth root of any number greater than 1.

The various steps of iteration will be as follows—

(1) Let the given number whose nth root to be found be A;
(2) Make $x_1 = 0$;
(3) Make $x_2 = A$;
(4) Find midpoint $x_m = (x_1 + x_2)/2$;
(5) COMPUTE $B = (x_m)^n$;
(6) If $B > A$ make $x_2 = x_m$ and GOTO step 4;
(7) If $B < A$ make $x_1 = x_m$ and GOTO step 4.
(8) CONTINUE the above till $B = A$ or the difference between A and B is too small, say 0.0001. Then x_m is the nth root of A to the desired accuracy.

EXAMPLE 17.3 Find the cube root of 15.625 by method of successive bisection.

Solution:

$$A = 15.625$$

ITERATION 1.

$$x_1 = 0$$
$$x_2 = A = 15.625$$
$$x_m = \frac{x_1 + x_2}{2} = (0 + 15.625)/2 = 7.8125$$
$$B = (x_m)^3 = 7.8125^3 > A.$$

$\therefore$ Make $x_2 = x_m = 7.8125$

ITERATION 2.

x_1 remains 0

$$x_2 = 7.8125$$
$$x_m = (x_1 + x_2)/2 = 3.90625$$
$$B = (x_m)^3 = 3.90625^3 > A.$$

$\therefore$ Make $x_2 = x_m = 3.90625$

ITERATION 3.

x_1 remains 0

$$x_2 = 3.90625$$
$$x_m = (x_1 + x_2)/2 = 1.953125$$
$$B = (x_m)^3 = (1.953125)^3 < A.$$

$\therefore$ Make $x_1 = x_m = 1.953125$

ITERATION 4.

$$\begin{aligned} x_1 &= 1.953125 \\ x_2 &\text{ remains } 3.90625 \\ x_m &= (x_1 + x_2)/2 = 2.9296875 \\ B &= (x_m)^3 = (2.9296875)^3 > A. \end{aligned}$$

$\therefore$ Make $x_2 = x_m = 2.9296875$

ITERATION 5.

$$\begin{aligned} x_1 &\text{ remains } 1.953125 \\ x_2 &= 2.9296875 \\ x_m &= (x_1 + x_2)/2 = 2.4414063 \\ B &= (x_m)^3 = (2.4414063)^3 < A. \end{aligned}$$

$\therefore$ Make $x_1 = x_m = 2.4414063$

ITERATION 6.

$$\begin{aligned} x_1 &= 2.4414063 \\ x_2 &\text{ remains } 2.9296875 \\ x_m &= (x_1 + x_2)/2 = 2.6855469 \\ B &= (x_m)^3 = (2.6855469)^3 > A. \end{aligned}$$

$\therefore$ Make $x_2 = x_m = 2.6855469$

ITERATION 7.

$$\begin{aligned} x_1 &\text{ remains } 2.4414063 \\ x_2 &= 2.6855469 \\ x_m &= (x_1 + x_2)/2 = 2.5634766 \\ B &= (x_m)^3 = (2.5634766)^3 > A. \end{aligned}$$

$\therefore$ Make $x_2 = x_m = 2.5634766$

ITERATION 8.

$$\begin{aligned} x_1 &\text{ remains } 2.4414063 \\ x_2 &= 2.5634766 \\ x_m &= (x_1 + x_2)/2 = 2.5024414 \\ B &= (x_m)^3 = (2.5024414)^3 = 15.67082 \approx A. \end{aligned}$$

So, the cube root of A is ≈ 2.501.

NOTE. The number of iterations in the above solution could be reduced if the initial values of x_1 and x_2 have been assumed 2 and 3, respectively. But, while writing a generalised program to find the nth root of any number, such assumptions of initial values in close range will not be practically possible. So, x_1 and x_2 have to be initially assumed as 0 and given numbers, respectively.

17.5.3 Regula Falsi Method

This method is also called method of FALSE POSITION or LINEAR INTERPOLATION or COMPUTED LINE APPROACH.

In this method also, the interval, say (x_1 and x_2), which contains a real root should be first ascertained, say, by incremental search method.

If x_3 is a root more accurate than x_1 and x_2, it is given as

$$x_3 = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}$$

The further steps in the procedure are

1. Find $f(x_3)$
2. Check whether it is ≈ 0
   If so GOTO step 7
3. Else check the signs of $f(x_1)$ and $f(x_3)$
4. If they are of opposite sign
   make $x_2 = x_3$ and compute new $f(x_2)$;
5. Compute x_3 using the above formula and GOTO step 1.
6. If they are of same sign, make $x_1 = x_3$ and compute new $f(x_1)$. GOTO step 5.
7. Display (ONE REAL root of given equation = x_3)
8. STOP

EXAMPLE 17.4 Solve the following equation by REGULA FALSI method: $x^2 - 2\cos x = 0$

Solution:

The equation is $x^2 - 2\cos x = 0$

$f(x) = x^2 - 2\cos x$

It is found that when $x = 1, f(x) = -0.0806$, and when $x = 2, f(x) = 4.8322$

Since, there is a change in sign the root lies between 1 and 2

ie,

$$\begin{aligned} x_1 &= 1 \qquad & f(x_1) &= -0.0806 \\ x_2 &= 2 \qquad & f(x_2) &= 4.8322 \end{aligned}$$

ITERATION 1. If x_3 = more accurate root

$$x_3 = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)} = \frac{1 \times 4.8322 - 2(-0.0806)}{4.8322 - (-0.0806)} = 1.0166718$$

$$f(x_3) = x_3^2 - 2\cos x_3 = -0.0187789 \neq 0$$

Also, it is of same sign as $f(x_1)$

So make

$x_1 = x_3 = 1.0166718$

New value of $f(x_1) = 1.0166718^2 - 2\cos(1.0166718 \text{ radians}) = -0.0187789.$

ITERATION 2.

$$x_3 = \frac{1.0166718 \times 4.8322 - 2(-0.0187789)}{4.8322 - (-0.0187789)} = 1.0206638$$

$$f(x_3) = x_3^2 - 2\cos x_3 = -0.003848 \neq 0$$

Also, it is of same sign as $f(x_1)$.

So make $x_1 = x_3 = 1.0206638$

New value of $f(x_1) = -0.003848$

ITERATION 3.

$$x_3 = \frac{1.0206638 \times 4.8322 - 2(-0.003848)}{4.8322 - (-0.003848)} = 1.0214448$$

$$f(x_3) = x_3^2 - 2\cos x_3 = -0.0009$$

Since, the absolute value of $f(x_3)$ is $< .001$ the iterations can be stopped taking that

$$f(x_3) \approx 0$$

So one root of the equation = latest value of $x_3 = 1.021448$ radians.

17.5.3.1 Derivation of x_3

Referring to Fig 17.4, the function $f(x)$ changes sign between values of x equal to x_1 and x_2. If x_3 is more accurate root than x_1 or x_2 it is derived as follows—

$$\frac{x_2 - x_3}{x_2 - x_1} = \frac{f(x_2)}{f(x_2) - f(x_1)}; \quad x_3 = x_2 - \frac{(x_2 - x_1)f(x_2)}{f(x_2) - f(x_1)}; \quad x_3 = \frac{x_1 f(x_2) - x_2(fx_1)}{f(x_2) - f(x_1)}$$

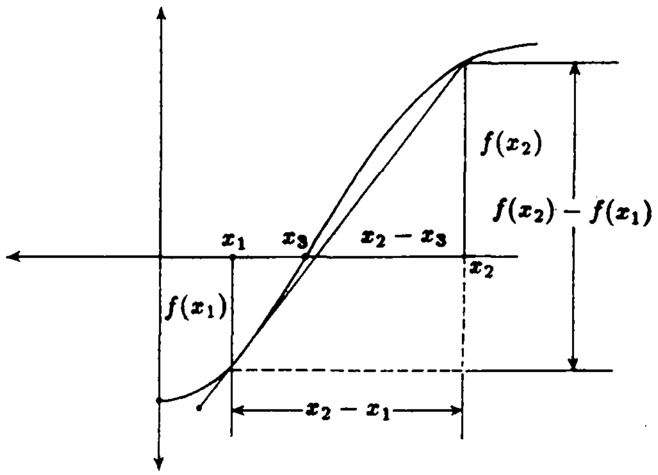


Fig 17.4

17.5.3.2 Algorithm: REGULA FALSI

(Given two values of x, namely x_1 and x_2 such that $f(x_1) * f(x_2) < 0$ and given ϵ (Epsilon) the desired accuracy, to find ROOT which is the value of x which satisfies $f(x) = 0$)

```
1. (Input values of $x_1, x_2$ and $\epsilon$)
   READ ($x_1, x_2, \epsilon$)
2. (Initialise)
   x-OLD $\leftarrow x_1$
   x-NEW $\leftarrow x_2$
3. (Compute $x_3$)
   $x_3 \leftarrow$ (x-OLD *f(x-NEW) −x NEW *f(x-OLD)/
   (f(x-NEW) −f(x-OLD))
4. (Check the convergence)
   IF $|f(x_3)| < E$ then GOTO 6
5. (ELSE)
   IF ( f(x-OLD) *f($x_3$)) < 0 THEN
   x-NEW $\leftarrow x_3$
   GOTO STEP 3
   ELSE x-OLD $\leftarrow x_3$
   GOTO STEP 3
6. ROOT $\leftarrow x_3$
7. (OUTPUT ROOT)
   WRITE ('ONE ROOT OF EQUATION=', ROOT)
8. (FINISHED)
   EXIT
```

17.5.4 SECANT METHOD

The SECANT METHOD is similar to the REGULA FALSI method, except that the two 'most recent' x values and their corresponding function values are used in obtaining a new approximation to the root.

In the REGULA FALSI method, whenever a new root x_3 is found using x_1 and x_2, it was checked whether $f(x_1)$ and $f(x_3)$ are of same sign. If so, x_1, was set to $x_3 (x_1 \leftarrow x_3)$ and with the new value of x_1 and existing value of x_2, the value of x_3 was computed. Else, x_2 was set to $x_3 (x_2 \leftarrow x_3)$ and with the existing value of x_1 and new value of x_2, the value of x_3 was computed.

In the SECANT METHOD both x_1 and x_2 will be modified after each iteration. x_1 will be set to x_2 or x-NEW ($x_1 \leftarrow x$-NEW) and x_2 or x-NEW will be set to x_3(x-NEW $\leftarrow x_3$) and the new value of x_3 will be found. The formula used will be the same as in REGULA FALSI method, namely,

$$x_3 = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}$$

17.5.4.1 Algorithm : Secant

(Given two values of x, namely x_1 and x_2, such that $f(x_1) * f(x_2) < 0$ and given ϵ (Epsilon) the desired accuracy, to find ROOT which is the value of x that satisfied $f(x) = 0$.

```
1. (Input values of $x_1, x_2$ and $\epsilon$)
   READ ($x_1, x_2, \epsilon$)
2. (INITIALISE)
   x-OLD ← $x_1$
   x-NEW ← $x_2$
3. (COMPUTE $X_3$)
   $x_3$ ← (x-OLD *f(x-NEW) −x-NEW *f(x-OLD))
4. (Check for convergence)
   IF $|f(x_3)| < \epsilon$ then GOTO 6
5. (ELSE)
   x-OLD ← x-NEW
   x-NEW ← $x_3$
   GOTO 3
6. ROOT ← $x_3$
7. (OUTPUT ROOT)
   WRITE ('ONE ROOT IS', ROOT)
8. (FINISHED)
   EXIT
```

Note that compared to REGULA FALSI method, there is no necessity in the SECANT method to check whether $f(x_3)$ is of same sign as $f(x_1)$ or $f(x_2)$. (refer Step—5 in both the algorithms)

EXAMPLE 17.5 Solve the equation under Example 17.4 by SECANT method.

Solution:

The equation is $x^2 - 2\cos x = 0$

$$f(x) = x^2 - 2\cos x$$

It is found when $x = 1$ $f(x) = -0.0806$

and when $x = 2$ $f(x) = 4.8322$

$x_1 = 1$ and $f(x_1) = -0.0806$

$x_2 = 2$ and $f(x_2) = 4.8322$

ITERATION 1.

$$x_3 = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)} = 1.0166718$$

(x_1, x_2, x_3 are in radians)

$$f(x_3) = x_3^2 - 2\cos x_3 = -0.0187789$$

Since, it is not ZERO set $x_1 \leftarrow x_2 = 2$

and $x_2 \leftarrow x_3 = 1.0166718$

ITERATION 2.

$$x_3 = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}$$

$x_1 = 2$ and $f(x_1) = 4.8322$

$x_2 = 1.0166718$ $f(x_2) = -.0187789$

$$\therefore \quad x_3 = \frac{2(-.0187789) - 1.0166718(4.8322)}{-.0187789 - 4.8322} = 1.0202682$$

$$f(x_3) = x_3^2 - 2\cos x_3 = -0.0036534$$

Since this is not 0

$$x_1 \leftarrow x_2 = 1.0166718$$

$$\text{and} \quad x_2 \leftarrow x_3 = 1.0202682$$

ITERATION 3.

$$f(x_1) = -0.187789$$

$$f(x_2) = -0.0036534$$

$$x_3 = \frac{1.016678(-0.0036534) - 1.0202682(-0.187789)}{-0.0036534 - (-0.187789)} = 1.020298$$

$$f(x_3) = x_3^2 - 2\cos x_3 = -0.0008992$$

(Since $f(x_3)$ is less than .001, the iterations may be stopped)

$x_3 = 1.020298$ radians (compare this value with that obtained by REGULA FALSI method)

EXAMPLE 17.6 Find a real root of the equations $x^3 - 2x - 5 = 0$ using secant method.

Solution:

$$f(x) = x^3 - 2x - 5$$

It is found that a real root lies between 2 and 3:

ie,

$$x_1 = 2 \qquad f(x_1) = -1$$

$$x_2 = 3 \qquad f(x_2) = 16$$

ITERATION 1.

$$x_3 = \frac{2(16) - 3(-1)}{16 - (-1)} = \frac{35}{17} = 2.0588235$$

$$f(x_3) = x_3^2 - 2x_3 - 5 = 0.3907941 \neq 0$$

So, set $x_1 = x_2 = 3$

and $x_2 = x_3 = 2.0588235$

$$f(x_1) = 16$$

$$f(x_2) = 0.3907941$$

ITERATION 2.

$$x_3 = \frac{3(0.3907941) - 2.0588235(16)}{0.3907941 - 16} = 2.0285212$$

$$f(x_3) = -0.61 \neq 0.$$

So set $x_1 = x_2 = 2.0588235$ and $x_2 = x_3 = 2.0285212$

$$f(x_1) = 0.3907941$$

$$f(x_2) = -0.61$$

ITERATION 3.

$$x_3 = \frac{2.0588235(-0.61) - 2.02852(0.3907941)}{-0.61 - 0.390794}$$
$$= 2.0486$$

(The student is advised to continue the iterations till convergence is obtained.)

17.4.5 Newton-Raphson Method

This is also called NEWTON method. This requires only one initial guess, ie, initial value of root, say x_1, of the given function $f(x)$.

A more accurate value of the root say x_2 is given by

$$x_2 = x_1 - f(x_1)/f'(x_1)$$

where $f'(x)$ is the derivative of $f(x)$. The derivation of this is as follows:

17.4.5.1 Derivation

Referring to Fig 17.5, x_1 is the initial guess value of the root-while x_a is the actual value. If x_2 is nearer to x_a than $x_1, \tan\theta = f(x_1)/(x_1 - x_2)$.

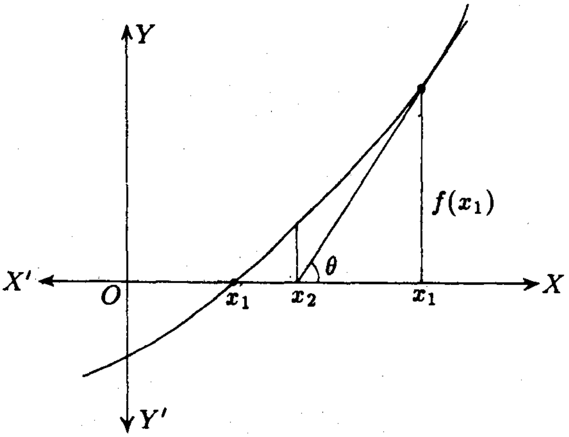


Fig 17.5

Since the derivative of the functions $f(x_1)$ say $f'(x_1)$ is equal to the slope of the tangent at x_1; ie, $\tan\theta = f'(x_1) = f(x_1)/x_1 - x_2)$

Hence $x_2 = x_1 - f(x_1)/f'(x_1)$

To find the root of an equations it is checked whether $f(x_2) \approx 0$ or less than EPSILON (a small quantity). If so, x_2 is taken as a ROOT. Else using the value of x_2 in place of x_1, its new value is calculated applying the above formula. The procedure is repeated until a convergence is come across.

17.4.5.2 Algorithm Newton

(Given x_1 an initial guess value of the root of $f(x)$ and the desired accuracy E, to find ROOT which is a root of $f(x)$)

```
1. (INPUT $x_1$ and $E$)
   READ $(x_1, E)$
2. (COMPUTE $x_2$)
   $x_2 \leftarrow x_1 - f(x_1)/f'(x_1)$
3. (check for convergence)
   IF $|f(x_2)| < E$ then GOTO 6
4. ELSE
   $x_1 \leftarrow x_2$
5. GOTO STEP 2
6. ROOT $\leftarrow x_2$
7. (OUTPUT ROOT)
   WRITE ('ONE ROOT OF THE EQUATION', ROOT)
8. (finished)
   EXIT
```

EXAMPLE 17.7 Find one of the roots of the transcendental equation $x^2 - 2\cos x = 0$ by NEWTON-RAPHSON method.

Solution:

(In most cases it is customary to assume the initial value of root as 1)

$$f(x) = x^2 - 2\cos x$$
$$f'(x) = 2x + 2\sin x$$

Assume $x_1 = 1$ radian.

ITERATION 1.

$$\begin{aligned} f(x_1) &= 1^2 - 2\cos(1) = -.0806 \\ f'(x_1) &= 2 \times 1 + 2\sin(1) = 3.6829 \\ x_2 &= x_1 - f(x_1)/f'(x_1) = 1 - \frac{-.0806}{3.6829} = 1.0218 \\ f(x_2) &= 1.0218^2 - 2\cos(1.0218) = 0 \end{aligned}$$

Hence root $= x_2 = 1.0218$ radians.

NOTE : The convergence of the value of the root can also be tested by comparing its values before and after each iteration, namely x_1 and x_2. If they are equal or the absolute value of the difference between them is too small (<Epsilon), then x_2 can be taken as the value of the root (refer example 17.6)

17.4.5.3 Corollary

The NEWTON-RAPHSON method can be used to find the nth root of any number A.

Putting $x^n = A$

$$f(x) = x^n - A = 0$$
$$f'(x) = nx^{n-1}$$

If x_1 = assumed approximate value of the root, and if x_2 = more accurate root

$$x_2 = x_1 - f(x_1)/f'(x_1)$$

The value of x_2 is successively improved until $x_2 \approx x_1$

EXAMPLE 17.8 Find the square root of 0.25 by Newton-Raphson method

$$\begin{aligned} x^2 &= 0.25 \\ f(x) &= x^2 - 0.25 = 0 \\ f'(x) &= 2x \\ x_1 &= 1 \end{aligned}$$

ITERATION 1.

$$\begin{aligned} f(x_1) &= 1^2 - 0.25 = 0.75 \\ f'(x_1) &= 2 \times 1 = 2 \\ x_2 &= x_1 - f(x_1)/f'(x_1) = 1 - \frac{0.75}{2} = 0.625 \end{aligned}$$

It is found that $x_2 \neq x_1$

$\therefore$ Make $x_1 = x_2 = 0.625$

ITERATION 2.

$$\begin{aligned} f(x_1) &= 0.625^2 - 0.25 = 0.1406 \\ f'(x_1) &= 2x_1 = 2 \times 0.625 = 1.25 \\ x_2 &= 0.625 - .1406/1.25 = 0.5125 \end{aligned}$$

It is found that $x_2 \neq x_1$

$\therefore$ Make $x_1 = x_2 = 0.5125$

ITERATION 3.

$$\begin{aligned} f(x_1) &= 0.5125^2 - 0.25 = 0.0126 \\ f'(x_1) &= 2 \times 0.5125 = 1.025 \\ x_2 &= 0.5125 - \frac{.0126}{1.025} = 0.5 \end{aligned}$$

It is found that $x_2 \neq x_1$

$\therefore$ Make $x_1 = x_2 = 0.5$

ITERATION 4.

$$\begin{aligned} f(x_1) &= 0.5^2 - 0.25 = 0 \\ f'(x_1) &= 2 \times 0.5 = 1 \\ x_2 &= 0.5 - 0/1 = 0.5 \end{aligned}$$

It is found that $x_2 = x_1$

$\therefore$ The square root of $0.25 = x_2 = 0.5$

17.6 CONCLUSION

Besides the above methods, the following are also available for solving algebraic or transcendental equations:

(i) Newton's second-order method;
(ii) The method of successive approximation;
(iii) MULLER's method;
(iv) The Quotient-difference method;
(v) GRAEFFE's root squaring method, and
(vi) BAIRSTOW's method.

These methods are left to those interested, for self-study.

Summary

For solving algebraic and transcendental equations, iterative methods have to be employed while using computers for the solution. Some of the iterative methods are (1) Incremental Search method (2) Method of Bisection (3) REGULA FALSI method (4) Secant method, and (5) Newton-Raphson method.

Exercise/Questions

17.1 Solve the following equations by Incremental search method:

(a) $x^3 - 9x^2 + 26x - 24 = 0$
(b) $e^x \cos x - 3 = 0$

17.2 Get any one root of the following equations using method of bisection:

(a) $x^3 - 4x^2 - x + 10 = 0$
(b) $\sqrt{x} - 4\tan(x/2) = 0$

Get the values of x_1 and x_2 between which $f(x)$ changes sign by incremental search method.

17.3 Solve the problems *a* and *b* in Exercise 17.2 by

(a) REGULA FALSI method, and
(b) SECANT method.

17.4 Solve by NEWTON-RAPHSON method

(a) $2\cos x - x^2 = 0$
(b) $x^3 + 4x^2 - 2x + 3 = 0$

17.5 Compute all the 3 roots of the equation

$$x^3 - 3x^2 + 3x - 4 = 0$$

[**HINT.** Obtain one root by Newton-Raphson method. Using it, reduce the equation to quadratic and get the other roots.]

17.6 Obtain the 4th root of 26.4 by

(a) Method of bisection, and
(b) by Newton-Raphson method.

17.7 Translate algorithms in this chapter into FORTRAN Programs.

for NOTES

Chapter Eighteen

Simultaneous Algebraic Equations—Matrix Inversion

18.1 INTRODUCTION

While solving technical problems like power system analysis, structural design etc the solutions will be of sets of simultaneous equations.

The simultaneous equations are of 2 major categories: (a) simultaneous algebraic equations, and (b) simultaneous differential equations.

The simultaneous algebraic equations alone are discussed here.

18.2 SIMULTANEOUS ALGEBRAIC EQUATIONS

These are of 2 types (i) Simultaneous linear algebraic equations, and (ii) Simultaneous non-linear algebraic equations.

The NON-LINEAR algebraic equations are of the form $f(x,y) = 0$, $g(x,y) = 0$ and their solutions involves Taylor series expansions and, as such, their solution is beyond the scope of this book.

18.2.1 Simultaneous Linear Algebraic Equations

These equations are come across in theory of Elasticity, electric-circuit analysis, vibrations, heat transfer etc These equations are of 2 types, namely (a) Homogeneous simultaneous linear algebraic equations, and (b) Non-homogeneous simultaneous linear algebraic equations. The homogeneous simultaneous equations will be of the form

$$\begin{aligned}
a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 + a_{1,3} \cdot x_3 &= 0 \\
a_{2,1} \cdot x_1 + a_{2,2} \cdot x_2 + a_{2,3} \cdot x_3 &= 0 \\
a_{3,1} \cdot x_1 + a_{3,2} \cdot x_2 + a_{3,3} \cdot x_3 &= 0
\end{aligned}$$

18.2.2 Non-Homogeneous Equations

If the constants to which each function is equated is not zero, it will be a set of non-homogeneous simultaneous linear algebraic equations.

eg,

$$\begin{aligned} a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 + a_{1,3} \cdot x_3 &= C_1 \\ a_{2,1} \cdot x_1 + a_{2,2} \cdot x_2 + a_{2,3} \cdot x_3 &= C_2 \\ a_{3,1} \cdot x_1 + a_{3,2} \cdot x_2 + a_{3,3} \cdot x_3 &= C_3 \end{aligned}$$

For solving the non-homogeneous simultaneous linear algebraic equations, they are to be recorded and stored in the matrix form. Sometimes the coefficients are stored in the form of a separate matrix which is called 'coefficient matrix' and the constants in another matrix which is in the form of a column matrix.

COEFFICIENT MATRIX is a matrix whose elements are the coefficients of the unknowns in the set of equations.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & \cdots & a_{3,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & \cdots & a_{n,n} \end{bmatrix} = \text{Coefficient matrix}$$

$$C = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \cdots \\ \cdots \\ C_n \end{bmatrix} = \text{COLUMN MATRIX}$$

If the constants are added to the coefficient matrix as the right-most element in each row, the matrix is called 'AUGMENTED MATRIX'. (refer matrix below)

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} & C_1 \\ a_{2,1} & a_{2,2} & \cdots & \cdots & a_{2,n} & C_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & \cdots & \cdots & a_{n,n} & C_n \end{bmatrix}$$

For formulating algorithms, it will be convenient in many cases if the constants are stored as an additional column of coefficients, (eg) the above matrix can be expressed as below

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} & a_{1,n+1} \\ a_{2,1} & a_{2,2} & \cdots & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & \cdots & \cdots & a_{n,n} & a_{n,n+1} \end{bmatrix}$$

18.2.3 Solution to Non-Homogeneous Simultaneous Equations

There are two types of solutions for the non-homogeneous simultaneous linear algebraic equations: (i) Direct methods, and (ii) Indirect methods (Iterative methods).

The direct methods are also known as ELIMINATION methods, or REDUCTION methods. The various elimination methods are: (i) Method using CRAMER's rule, (ii) GAUSS Elimination method, (iii) GAUSS-JORDAN Elimination method, (iv) CROUT's (CHOLESKY's) method, and (v) Matrix Inversion method.

The Cramer's rule method involves eliminations of unknowns by combining equations and the use of determinants. It is used only when three simultaneous equations are to be solved. If the number of equations are more, this method of solution will involve a number of multiplications and hence will be time-consuming. As such, the other elimination methods alone are used in practice. In this chapter, the Gauss elimination method, the Gauss Jordan method and the matrix inversion methods are discussed.

18.2.4 Solution by Gauss Elimination

In this method, a set of n equations in n unknowns is reduced to an equivalent triangular set which is then solved by back substitution. (An equivalent set is a set having identical solution values.) The Example 18.1 illustrates the above process.

EXAMPLE 18.1 Solve:

$$\begin{aligned} 2x_1 + 5x_2 + 2x_3 &= 6 \\ 2x_1 + x_2 + 5x_3 &= 1 \\ 3x_1 - 2x_2 + 4x_3 &= -9 \end{aligned}$$

by Gauss Elimination method.

Solution:

Step 1. (Write down the augmented matrix)

$$\begin{bmatrix} 2 & 5 & 2 & 6 \\ 2 & 1 & 5 & 1 \\ 3 & -2 & 4 & -9 \end{bmatrix}$$

Step 2. (Divide the first row by its PIVOT. PIVOT is the element in the main diagonal of the matrix; its row index and column index will be the same).

$$\begin{bmatrix} 1 & {}^5/_2 & 1 & 3 \\ 2 & 1 & 5 & 1 \\ 3 & -2 & 4 & -9 \end{bmatrix}$$

(The above division is to generate the value of unity in the first diagonal position)

Step 3. (Eliminate x_1 from 2nd row. This is done by multiplying first row by the first element in the second row and subtracting the product from the second row) ($R_2 \leftarrow R_2 - 2 * R_1$)

$$\begin{bmatrix} 1 & {}^5/_2 & 1 & 3 \\ 0 & -4 & 3 & -5 \\ 3 & -2 & 4 & -9 \end{bmatrix}$$

Step 4. (Eliminate x_1 from row 3 by a similar process) : $R_3 \leftarrow R_3 - 3 * R_1$

$$\begin{bmatrix} 1 & {}^{5}/_{2} & 1 & 3 \\ 0 & -4 & 3 & -5 \\ 0 & {}^{-19}/_{2} & 1 & -18 \end{bmatrix}$$

Step 5. (Divide the second row by its pivot) PIVOT is an element whose row index equals column index

$$\begin{bmatrix} 1 & {}^{5}/_{2} & 1 & 3 \\ 0 & 1 & {}^{-3}/_{4} & {}^{5}/_{4} \\ 0 & {}^{-19}/_{2} & 1 & -18 \end{bmatrix}$$

Step 6. (Eliminate x_2 from row 3) : $R_3 \leftarrow R_3 - \left(\frac{-19}{2} * R_2\right)$

$$\begin{bmatrix} 1 & {}^{5}/_{2} & 1 & 3 \\ 0 & 1 & {}^{-3}/_{4} & {}^{5}/_{4} \\ 0 & 0 & {}^{-49}/_{8} & {}^{-49}/_{8} \end{bmatrix}$$

Step 7. (Divide the third row by its pivot)

$$\begin{bmatrix} 1 & {}^{5}/_{2} & 1 & 3 \\ 0 & 1 & {}^{-3}/_{4} & {}^{5}/_{4} \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The above steps fall under the process of 'elimination'.

The following steps are under 'back-substitution':

Step 8. Write down the equation from the above triangular matrix. From the last row, obtain value of x_3

$$x_1 + \frac{5}{2}x_2 + x_3 = 3; \quad x_2 - \frac{3x_3}{4} = \frac{5}{4}; \quad x_3 = 1.$$

Step 9. Substitute value of x_3 in row-2 and obtain the value of x_2:

$$x_2 = \frac{5}{4} + \frac{3x_3}{4} = \frac{5}{4} + \frac{3}{4} = 2.$$

Step 10. Substitute the values of x_3 and x_2 in row-1 and obtain the value of x_1 :

$$x_1 = 3 - \frac{5}{2}x_2 - x_3 = 3 - \frac{5}{2} \times 2 - 1 = -3.$$

Step 11. (Output results)

$$x_1 = -3, \quad x_2 = 2, \quad x_3 = 1.$$

18.2.4.1 Pivoting

In the above example, it so happens that all the variables are of integer values. But, in many cases, they will be all real and rounding has to be adopted while finalising the results. The round-off errors can be reduced by 'pivoting'. There are 2 types of pivoting, namely (a) partial pivoting, and (b) pivoting or full pivoting.

Partial pivoting involves interchange of two rows such that the element with the largest absolute magnitude becomes the pivot element.

The given set of simultaneous equations will be in the following order after partial pivoting:

$$\begin{aligned} 3x_1 - 2x_2 + 4x_3 &= -9 \\ 2x_1 + 5x_2 + 2x_3 &= 6 \\ 4x_1 + x_2 + 5x_3 &= 1 \end{aligned}$$

When both rows and columns are interchanged, it is called full-pivoting (pivoting).

Partial pivoting involves simple change in the order of the equations. If columns are interchanged, the order of variables within the equations will be changed and one must keep track of the order the variables will occur in the final phase of calculations. In practice, only partial pivoting (interchange of rows) is adopted. After partial pivoting, the Gauss elimination method is used to get the result.

Another benefit derived in partial or full pivoting is, if the pivotal element in any row is zero, it gets interchanged with a row with a non-zero pivot. This prevents any possible divisions by zero.

18.2.4.2 Algorithm Gauss

(Given a set of simultaneous equations in the form of an augmented matrix, the algorithm gives the solution, partial pivoting is done before the elimination process)

```
1.  (Input elements of augmented matrix of $n$ rows and $m$ columns where $m = n + 1$)
    Repeat FOR I = 1,2,3......N
    Repeat FOR J = 1,2,3......M
    READ (A(I,J))
2.  (Perform partial pivoting)
    CALL PIVOT (A,N,M)
    (The sub-algorithm PIVOT is used for pivoting)
3.  Repeat THRU STEP 7 FOR I = 1,2,3...N (perform elimination)
4.  Repeat THRU STEP 7 FOR J = I,I+1,I+2,......M
5.  A(I,J) ← A(I,J)/A(I,I)
6.  Repeat THRU STEP 7 for k = I+1,I+2,......N.
7.  Repeat for J = I,I+1,......M.
    A(K,J) ← A(K,J) − A(K,J) * A(I,J)
8.  (Compute X(N))
    X(N) ← A(N,M)
9.  (Perform back-substitution)
    Repeat THRU STEP 12 for I = N−1,N−2,N−3,......,1
10. SUM ← 0
11. Repeat for J = I+1,I+2,......N
    SUM ← SUM +A(I,J) * X(J)
12. x(I) ← A(I,M)− SUM
13. (OUTPUT RESULTS)
    Repeat for I = 1,2,3......N
    WRITE (X(I))
```

```
14. (finished)
    EXIT.
```

The above algorithm uses the subalgorithm PIVOTT for partial pivoting. The subalgorithm is given below.

18.2.4.3 Sub-Algorithm PIVOTT (A,N,M)

(This sub-algorithm performs partial-pivoting by interchanging rows on a matrix of N rows and M columns, where $M = N + 1$)

```
1. REPEAT thru STEP 8 FOR I = 1, 2, 3 ...... N
2. BIG ← ABS (A(I,I))
3. L ← I
4. K ← I + 1
5. Repeat thru STEP 7 for J = K, K + 1 ...... N
6. B ← ABS (A(J,I))
7. IF B > BIG then
       BIG ← B
       L ← J
8. IF L ≠ I Then (exchange)
       Repeat for J = 1, 2, 3, ...... M
           A(L,J) ↔ A(I,J)
9. RETURN.
```

18.2.5 Gauss-Jordan Method

This is an improvisation on the Gauss elimination method. The elimination process is of 2 steps.

Step 1. (FORWARD ELIMINATION) Starting from the top row, successively divide each row by its diagonal element (PIVOT), leaving a 1 on the diagonal and subtract a multiple of that row from each subsequent row, leaving zeros in the columns below the main diagonal.

Step 2. BACK ELIMINATION or REVERSE ELIMINATION—Subtract multiples of lower rows from the higher rows so that all the elements above the main diagonal are reduced to 0.

Operation of these 2 steps on the augmented matrix will result 1's in the diagonal elements as shown below—

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 & a_{1,n+1} \\ 0 & 1 & 0 & \cdots & \cdots & 0 & a_{2,n+1} \\ 0 & 0 & 1 & \cdots & \cdots & 0 & a_{3,n+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \cdots & 1 & a_{n,n+1} \end{bmatrix}$$

From this, the values of variables $x_1, x_2, \ldots, x_n$ can be obtained easily from the corresponding rows:

$$x_1 = a_{1,n+1}$$
$$x_2 = a_{2,n+1} \text{ etc}$$
$$x_n = a_{n,n+1}$$

In this case also, PIVOTING may be done before the elimination process.

EXAMPLE 18.2 Solve the simultaneous equations given under Example 18.1 by GAUSS-JORDAN method.

Solution:

The given equations are

$$\begin{aligned} 2x_1 + 5x_2 + 2x_3 &= 6 \\ 2x_1 + x_2 + 5x_3 &= 1 \\ 3x_1 - 2x_2 + 4x_3 &= -9 \end{aligned}$$

Step 1. (Perform partial pivoting)

The order of equations will now be

$$\begin{aligned} 3x_1 - 2x_2 + 4x_3 &= -9 \\ 2x_1 + 5x_2 + 2x_3 &= 6 \\ 2x_1 + x_2 + 5x_3 &= 1 \end{aligned}$$

Step 2. Write down the 'Augmented Matrix'

$$\begin{bmatrix} 3 & -2 & 4 & -9 \\ 2 & 5 & 2 & 6 \\ 2 & 1 & 5 & 1 \end{bmatrix}$$

Step 3. Divide the first row by its pivot

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & {}^{4}/_{3} & -3 \\ 2 & 5 & 2 & 6 \\ 2 & 1 & 5 & 1 \end{bmatrix}$$

Step 4. [Eliminate x_1 from row 2

$R_2 \leftarrow R_2 - 2 * R_1$]

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & {}^{4}/_{3} & -3 \\ 0 & {}^{-19}/_{3} & {}^{-2}/_{3} & 12 \\ 2 & 1 & 5 & 1 \end{bmatrix}$$

Step 5. [Eliminate x_1 from row 3

$R_3 \leftarrow R_3 - 2 * R_1$]

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & {}^{4}/_{3} & -3 \\ 0 & {}^{-19}/_{3} & {}^{-2}/_{3} & 12 \\ 0 & {}^{7}/_{3} & {}^{7}/_{3} & 7 \end{bmatrix}$$

Step 6. (Divide the second row by its pivot)

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & {}^{4}/_{3} & -3 \\ 0 & 1 & {}^{-2}/_{19} & {}^{36}/_{19} \\ 0 & 0 & {}^{147}/_{57} & {}^{49}/_{19} \end{bmatrix}$$

Step 7. [Eliminate x_2 from row 3

$R_3 \leftarrow R_3 - 7/3 * R_2$]

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & {}^{4}/_{3} & -3 \\ 0 & 1 & {}^{-2}/_{19} & {}^{36}/_{19} \\ 0 & 0 & {}^{147}/_{57} & {}^{49}/_{19} \end{bmatrix}$$

Step 8. (Divide the third row by its pivot)

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & {}^{4}/_{3} & -3 \\ 0 & 1 & {}^{-2}/_{19} & {}^{36}/_{19} \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

By the above steps of forward elimination, the elements below the main diagonal have been reduced to zero.

Similarly, by a process of 'elimination backwards', the elements above the main diagonal will now be reduced to zero.

Step 9. [Eliminate x_3 from row 2

$R_2 \leftarrow R_2 - (-2/9) \times R_3$]

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & {}^{4}/_{3} & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Step 10. [Eliminate x_3 from row 1

$R_1 \leftarrow R_1 - \frac{4}{3} * R_3$]

$$\begin{bmatrix} 1 & {}^{-2}/_{3} & 0 & {}^{-13}/_{3} \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Step 11. [Eliminate x_2 from row 1

$R_1 \leftarrow R_1 - (\frac{-2}{3}) * R_2$]

$$\begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Step 12. Rewriting the above, as equations

$$\begin{aligned} x_1 + 0x_2 + 0x_3 &= -3 \\ 0x_1 + x_2 + 0x_3 &= 2 \\ 0x_1 + 0x_2 + x_3 &= 1 \\ \text{ie,} \quad x_1 &= -3 \\ x_2 &= 2 \\ x_3 &= 1 \end{aligned}$$

The disadvantage in this method is that it requires more Arithmetic operations. But the advantage over the Gauss method is, no backward substitution is required to get the values of variables one by one.

18.2.5.1 Algorithm Gauss-Jordan

(Given a set of simultaneous equations in the form of an augmented matrix, the algorithm gives the solution. Partial pivoting is done before the elimination process)

```
    (Steps 1 to 7, same as in Algorithm Gauss)
 8. (Perform backward elimination)
    Repeat then Step 10 for I = n, n - 1, n - 2, ...... 2
 9. Repeat for J = 1, 2, ...... I - 1
    (Reduce the rightmost column elements in each row)
    A(J, M) ← A(J, M) - A(I, M) * A(J, I)
    (EQUATE other elements except the pivotal elements to ZERO)
    A(J, I) ← 0
10. (OUTPUT THE RESULTS)
    Repeat for I = 1, 2, 3, ...... N
    WRITE ('x(',I,')=', A(I,M))
11. (FINISHED)
    EXIT.
```

18.2.6 Matrix-Inversion Method

A given set of simultaneous equations will be of the form

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots\ldots + a_{1,n}x_n &= C_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots\ldots + a_{2,n}x_n &= C_2 \\ &\text{etc} \\ a_{n,1}x_1 + a_{n,2}x_2 + \ldots\ldots + a_{n,n}x_n &= C_n \end{aligned}$$

This can be written as

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & \cdots & \cdots & a_{n,n} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ \cdots \\ C_n \end{bmatrix}$$

or, $A * x = C$

ie, $x = \frac{C}{A} = C * A^{-1}$

where A^{-1} is called the Inverse of the square matrix A.

The inverse A^{-1} of a matrix A is defined to be such that.

$A * A^{-1} = A^{-1} * A = I$ where I is called the UNIT matrix or IDENTITY matrix. It will have 1's along its main diagonal and its other elements will be zeros.

The matrix-inversion method of solving simultaneous equations will be to obtain the inverse A^{-1} of the coefficient matrix A and multiply this inverse matrix with the column matrix of the constants. The result will be a column matrix giving the values of the variables x_1, x_2 etc ... x_n.

18.2.6.1 Matrix Inversion

The inverse of a matrix can be obtained by any one of the following methods (1) by using COFACTORS, (2) by using CHARACTERISTIC equations, and (3) by using IDENTITY matrix.

Of these, the method by using IDENTITY matrix is best suited for computerisation.

This method will consist of : (i) Set up an IDENTITY matrix of same size as the coefficient matrix A which is of size $N \times N$ (square matrix), (ii) Append the IDENTITY matrix to A rowwise, (iii) Reduce A, using the procedure in Gauss-Jordon method, to an IDENTITY matrix. Each manipulation done on A is also performed on the appended unit matrix.

When finally A reduces to a unit (IDENTITY) matrix, the unit matrix will bc transformed into the inverse of A, namely A^{-1}. The inverse A^{-1} is removed and stored separately. As stated earlier, the product of this inverse and the column matrix of constants will give the values of the variables.

EXAMPLE 18.3 Solve the following system by matrix inversion method

$$\begin{aligned} 2x_1 + x_2 + x_3 &= 9 \\ 3x_1 + 2x_2 + 3x_3 &= 16 \\ x_1 + 4x_2 + 9x_3 &= 20 \end{aligned}$$

Solution:

Step 1. Write down the coefficient matrix A and the constant matrix C

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 3 & 2 & 3 \\ 1 & 4 & 9 \end{bmatrix} \quad C = \begin{bmatrix} 9 \\ 16 \\ 20 \end{bmatrix}$$

Step 2. Append IDENTITY MATRIX to A to get the augmented matrix

$$\left[\begin{array}{ccc|ccc} 2 & 1 & 1 & 1 & 0 & 0 \\ 3 & 2 & 3 & 0 & 1 & 0 \\ 1 & 4 & 9 & 0 & 0 & 1 \end{array}\right]$$

(The further steps will be as in Gauss-Jordan method)

Step 3. $R_1 \leftarrow R_1/2$

$$\left[\begin{array}{ccc|ccc} 1 & {}^1/_2 & {}^1/_2 & {}^1/_2 & 0 & 0 \\ 3 & 2 & 3 & 0 & 1 & 0 \\ 1 & 4 & 9 & 0 & 0 & 1 \end{array}\right]$$

Step 4. $R_2 \leftarrow R_2 - 3R_1$

$$\left[\begin{array}{ccc|ccc} 1 & {}^1/_2 & {}^1/_2 & {}^1/_2 & 0 & 0 \\ 0 & {}^1/_2 & {}^3/_2 & -{}^3/_2 & 1 & 0 \\ 1 & 4 & 9 & 0 & 0 & 1 \end{array}\right]$$

Step 5. $R_3 \leftarrow R_3 - 1 * R_1$

$$\left[\begin{array}{ccc|ccc} 1 & {}^1/_2 & {}^1/_2 & {}^1/_2 & 0 & 0 \\ 0 & {}^1/_2 & {}^3/_2 & -{}^3/_2 & 1 & 0 \\ 0 & {}^7/_2 & {}^{17}/_2 & -{}^1/_2 & 0 & 1 \end{array}\right]$$

Step 6. $R_2 \leftarrow R_2/\frac{1}{2}$

$$\left[\begin{array}{ccc|ccc} 1 & {}^1/_2 & {}^1/_2 & {}^1/_2 & 0 & 0 \\ 0 & 1 & 3 & -3 & 2 & 0 \\ 0 & {}^7/_2 & {}^{17}/_2 & -{}^1/_2 & 0 & 1 \end{array}\right]$$

Step 7. $R_3 \leftarrow R_3 - \frac{7}{2} * R_2$

$$\left[\begin{array}{ccc|ccc} 1 & {}^{1}/_{2} & {}^{1}/_{2} & {}^{1}/_{2} & 0 & 0 \\ 0 & 1 & 3 & -3 & 2 & 0 \\ 0 & 0 & -2 & 10 & -7 & 1 \end{array}\right]$$

Step 8. $R_3 \leftarrow R_3/(-2)$

$$\left[\begin{array}{ccc|ccc} 1 & {}^{1}/_{2} & {}^{1}/_{2} & {}^{1}/_{2} & 0 & 0 \\ 0 & 1 & 3 & -3 & 2 & 0 \\ 0 & 0 & 1 & -5 & {}^{7}/_{2} & -{}^{1}/_{2} \end{array}\right]$$

Step 9. $R_2 \leftarrow R_2 - 3 * R_3$

$$\left[\begin{array}{ccc|ccc} 1 & {}^{1}/_{2} & {}^{1}/_{2} & {}^{1}/_{2} & 0 & 0 \\ 0 & 1 & 0 & 12 & {}^{17}/_{2} & {}^{3}/_{2} \\ 0 & 0 & 1 & 10 & -7 & 1 \end{array}\right]$$

Step 10. $R_1 \leftarrow R_1 - R_3 * \frac{1}{2}$

$$\left[\begin{array}{ccc|ccc} 1 & {}^{1}/_{2} & 0 & 3 & {}^{-7}/_{4} & {}^{1}/_{4} \\ 0 & 1 & 0 & 12 & {}^{-17}/_{2} & {}^{3}/_{2} \\ 0 & 0 & 1 & -5 & {}^{7}/_{2} & {}^{-1}/_{2} \end{array}\right]$$

Step 11. $R_1 \leftarrow R_1 - R_2 * \frac{1}{2}$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -3 & {}^{5}/_{2} & {}^{-1}/_{2} \\ 0 & 1 & 0 & 12 & {}^{-17}/_{2} & {}^{3}/_{2} \\ 0 & 0 & 1 & -5 & {}^{7}/_{2} & {}^{-1}/_{2} \end{array}\right]$$

By now, the coefficient matrix is reduced to an IDENTITY matrix. The other section of the above will be the inverse (A^{-1}) of the given coefficient matrix A.

Step 12. Extract the inverse matrix A^{-1}

$$A^{-1} = \begin{bmatrix} -3 & {}^{5}/_{2} & {}^{-1}/_{2} \\ 12 & {}^{-17}/_{2} & {}^{3}/_{2} \\ -5 & {}^{7}/_{2} & {}^{-1}/_{2} \end{bmatrix}$$

Step 13. Multiply A^{-1} and the column matrix of constant C, to get values of x_1, x_2 and x_3.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = A^{-1} * C = \begin{bmatrix} -3 & {}^{5}/_{2} & {}^{-1}/_{2} \\ 12 & {}^{-17}/_{2} & {}^{3}/_{2} \\ -5 & {}^{7}/_{2} & {}^{-1}/_{2} \end{bmatrix} * \begin{bmatrix} 9 \\ 16 \\ 20 \end{bmatrix}$$

$$x_1 = -3 \times 9 + 5/2 \times 6 - 1/2 \times 20 = 3$$

$$x_2 = 12 \times 9 - 17/2 \times 16 + 3/2 \times 20 = 2$$

$$x_3 = -5 \times 9 + 7/2 \times 16 - 1/2 \times 20 = 1$$

18.2.6.2 Sub-Algorithm Invert-Mat (A,N,N, AINV)

(Given a square matrix A of size $N \times N$, this algorithm gets its inverse $AINV$ using the IDENTITY MATRIX)

```
Step 1. (Constract the coefficient matrix)
    Repeat for $I = 1, 2, 3 \cdot \ldots N$
    Repeat for $J = 1, 2, 3 \cdot \cdot N$
    $B(I, J) \leftarrow A(I, J)$
Step 2. (Append the IDENTITY matrix to the coefficient matrix)
    Repeat for $I = 1, 2, 3 \ldots\ldots N$
    Repeat for $J = N + 1, N + 2 \ldots\ldots 2N$
    IF $(J = N + I)$ then $B(I, J) \leftarrow 1$
    else $B(I, J) \leftarrow 0$
Step 3. (Perform forward elimination, Repeat thru Step 7 for $I = 1, 2, 3 \ldots\ldots N$
Step 4. Repeat thru Step 7 for $J = I, I + 1, I + 2 \ldots\ldots 2N$
Step 5. $B(I, J) \leftarrow B(I, J)/B(I, I)$
Step 6. Repeat thru Step 7 for $K = I + 1, I + 2, \ldots\ldots N$
Step 7. Repeat for $J = I, I + 1, I + 2, \ldots\ldots 2N$
    $B(K, J) \leftarrow B(K, J) - B(K, J) * B(I, J)$
Step 8. (Perform backward elimination) Repeat thru Step 9 for $I = N - 1, N - 2, \ldots\ldots 1.$
    $K \leftarrow I + 1$
Step 9. Repeat for $J = I + 1, I + 2, \ldots\ldots 2N$
    $A(I, J) \leftarrow A(I, J) - A(I, J) * A(K, K)$
Step 10. Output the inverse matrix $AINV$
    Repeat for $I = 1, 2 \ldots\ldots N$
    Repeat for $J = N + 1, N + 2, \ldots\ldots 2N$
    $K \leftarrow J - N$
    $AINV(I, K) \leftarrow B(I, J)$
Step 11. RETURN
```

18.2.6.3 Algorithm Matinvn

(Given a set of simultaneous equations in the form of a coefficient matrix of size $N \times N$ and a constant matrix of size $N \times 1$ to find the values of n variables $x_1, x_2 \ldots x_n$.

```
1. (Input elements of coefficient matrix)
   Repeat for I = 1, 2, 3 ...... N
   Repeat for J = 1, 2, 3 ...... N
   READ (A(I, J))
2. (Input the constant matrix of single column)
   Repeat for I = 1, 2, 3 ... N
   Read (C(I))
3. (Compute the inverse of coefficient matrix by using the subalgorithm INVERT-MAT)
   CALL INVERT-MAT (A, N, N, AINV)
```

```
4. (Multiply matrix AINV with constant matrix to get the solution vector)
   Repeat for I = 1,2,3......N
   x(I) ← 0
   Repeat for J = 1,2,3......N
   X(I) ← X(I) + AINV(I,J) * C(J)
5. (Output the solution vector $x_1$ to $x_m$)
   Repeat for I = 1,2,3......N
   WRITE (x(I))
6. (Finished)
   EXIT
```

The direct methods discussed above are not suitable for solving very large matrices since more and more multiplications and subtractions have to be performed as the number of equations increases. Also, the round-off error may be considerable. In such cases, indirect methods are preferable.

18.3 INDIRECT METHODS OF SOLUTION TO NON-HOMOGENEOUS LINEAR ALGEBRAIC EQUATIONS

The indirect methods are also called ITERATIVE methods. These have some advantages over the direct methods discussed so far, especially when the coefficient matrix is sparse, ie, when it contains a number of zero elements.

There are two iterative methods, namely (1) JACOBI method, and (2) GAUSS-SEIDEL method. Of these, the Gauss-Seidel is found to yield convergence of result somewhat faster than the other and hence it is discussed below.

18.3.1 GAUSS-SEIDEL Method

As stated earlier, this method finds the solution to a set of equtions by an iterative technique. An initial approximate value is assumed for each variable. It is repeatedly refined until a value close to the solution is obtained. Since each approximation depends on only the previous approximation, round-off error does not accumulate. Also, the method is applicable to non-linear equations.

In this case, partial pivoting may be done before solving the equations. This will result in faster convergence of results as also less round-off errors.

EXAMPLE 18.4 Solve the following equations by Gauss-Seidel method:

$$\begin{aligned} 3x_1 + 20x_2 - x_3 &= -18 \\ 2x_1 - 3x_2 + 20x_3 &= 25 \\ 20x_1 + x_2 - 2x_3 &= 17 \end{aligned}$$

Solution:

Step 1. Perform partial pivoting

$$\begin{aligned} 20x_1 + x_2 - 2x_3 &= 17 \\ 3x_1 + 20x_2 - x_3 &= -18 \\ 2x_1 - 3x_2 + 20x_3 &= 25 \end{aligned}$$

Step 2. (Solve equation 1 for x_1, equation 2 for x_2 and so on)

$$x_1 = (17 - x_2 + 2x_3)/20 \quad (1)$$

$$x_2 = (-18 - 3x_1 + x_3)/20 \quad (2)$$

$$x_3 = (25 - 2x_1 + 3x_2)/20 \quad (3)$$

Step 3. (Assume initial values for x_1, x_2, x_3. In many cases, the value assumed is ZERO).

$$x_1 = 0$$
$$x_2 = 0$$
$$x_3 = 0$$

Step 4. (ITERATION 1) x-OLD $\leftarrow x_1 = 0$

Step 5. (Using the most recent values of x_2 and x_3, evaluate x_1 from equation 1)

$$x_1 = (17 - 0 + 2 * 0)/20 = 0.85$$

Step 6. (Using the most recent values of x_1 and x_3, evaluate x_2 from equation 2)

$$x_2 = (-18 - 3 * 0.85 + 0)/20 = -1.0275$$

Step 7. (Using the most recent values of x_1 and x_2, evaluate x_3 from equation 3)

$$x_3 = (25 - 2 * 0.85 + 3 * (-1.0275))/20 = 1.0109$$

Step 8. Compare x_1 and x-OLD. Since they are not equal or since the absolute value of the difference is not small, set x-OLD $\leftarrow x_1 = 0.85$ (and continue the iterations).

Step 9. (ITERATION 2)

(Using the most recent values of x_2 and x_3, evaluate x_1 from equation 1)

$$x_1 = (17 - (1 * (-1.0275)) + 2 \times 1.0109)/20 = 1.0025$$

Step 10. (Using the most recent values of x_1 and x_3, evaluate x_2 from equation 2)

$$x_2 = (-18 - 3 * 1.0025 + 1.0109)/20 = -0.9998$$

Step 11. (Using the most recent values of x_1 and x_2, evaluate x_3 from equation 3)

$$x_3 = (25 - 2 * 1.0025 + 3 * (-0.9998))/20 = 0.9998$$

Step 12. Compare x_1 and x-OLD. They are not equal. So, set x-OLD $\leftarrow x_1 = 1.0025$

Step 13. (ITERATION 3) (Compute x_1 from equation 1)

$$x_1 = (17 - (-0.9998) + 2 * 0.9998)/20 = 1.0$$

Step 14. (Compute x_2 from equation 2)

$$x_2 = (-18 - 3 * 1 + 0.9998)/20 = -1.0$$

Step 15. (Compute x_3 from equation 3)

$$x_3 = (25 - 2 * 1 + 3 * (-1))/20 = 1.0$$

Step 16. (Check if $x_1 = x$-OLD)

Since they are not equal, set x-OLD $= x_1 = 1.0$

Step 17. (ITERATION 4) (Compute x_1 from equation 1)

$$x_1 = (17 - (-1) + 2 * 1)/20 = 1.0$$

Step 18. (Compute x_2 from equation 2)

$$x_2 = (-18 - 3 * 1 + 1)/20 = -1.0$$

Step 19. (Compute x_3 from equation 3)

$$x_3 = (25 - 2 * 1 + 3 * (-1))/20 = 1.00$$

Step 20. (Compare x_1 and x-OLD) (Since they are equal, stop the iteration and output the latest values)

$$x_1 = 1.0$$
$$x_2 = -1.0$$
$$x_3 = 1.0$$

Sometimes the algorithm may be terminated after specified number of iterations instead of when the values of x_1 for two consecutive iterations are within a specified accuracy.

18.3.1.1 Algorithm Gauss-Seidel

(Given a set of simultaneous equations with n unknowns and a set of initial guess values $x_1, x_2, \ldots\ldots x_n$ as also the desired accuracy E (epsilon), to solve the equations by iterative method)

```
1.  (Input the augmented matrix)
    M ← N + 1
    Repeat for I = 1, 2, 3, ...... N
    Repeat for J = 1, 2, 3, ...... M
    READ (A(I, J))
2.  (Input the initial guess values)
    Repeat for I = 1, 2, 3 ...... N
    READ (X(I))
3.  (Inpute values of desired accuracy and maximum number of iterations to be performed)
    READ (ε, MAXITN)
4.  (INITIALISE)
    NITN ← 0
5.  X-OLD ← x₁
6.  (Perform iterations)
    Repeat thru Step 9 for I = 1, 2, 3 ...... N
7.  SUM ← 0
8.  Repeat for J = 1, 2, 3 ...... N
    If I ≠ J then
    SUM ← SUM + A(I, J) * X(J)
9.  X(I) ← (A(I, M) − SUM)/A(I, I)
10. (Check for convergence)
    If (ABS(x₁ − x-OLD) < ε) Then WRITE ('THERE IS CONVERGENCE; THE VALUES ARE
    AS BELOW')
    Go To 14
11. (Else) (Add 1 to NITN)
    NITN ← NITN+1
```

```
12. (Check if number of iterations have reached max).
    If NITN = MAXITN then WRITE ('NO CONVERGENCE SO FAR. The final values of
    variables are given below')
    GOTO 14
13. (ELSE) (IF NITN<MAXITN)
    GOTO 5
14. (OUTPUT the VALUES)
    Repeat for I = 1,2,3,......N
    WRITE (x(I))
15. (Finished)
    EXIT.
```

Summary

While solving technical problems, simultaneous linear algebraic equations are come across. These are of two types, (i) homogenous equations, and (ii) non-homogeneous equations.

For solving non-homogeneous equations using computers, there are 2 types of numercial methods—(1) Direct methods, and (2) Indirect methods.

The three numerical methods falling under the direct methods are (i) Gauss elimination method, (2) Gauss-Jordan method, and (3) Matrix Inversion method. These are also called 'Elimination methods'.

The indirect methods are also known as Iterative methods. The iterative method which is mostly adopted is the Gauss-Seidel method.

Exercise/Questions

18.1 Differentiate between Homogeneous and Non-homogeneous linear algebraic equations.

18.2 Solve the following set of equations and give the values of x_1, x_2 and x_3 to the accuracy of 4 decimal places—

The solution is to be obtained by

(1) GAUSS-SEIDEL METHOD
(2) GAUSS-JORDAN METHOD
(3) GAUSS-ELIMINATION METHOD and
(4) MATRIX-INVERSION METHOD

The results are to be compared.

(Adopt Partial pivoting if necessary and possible)

$$\begin{aligned}7.000x_1 + 15.262x_2 + 43.173x_3 &= 13.0643\\43.1754x_1 + 136.4352x_2 + 454.116x_3 &= 90.762\\15.2620x_1 + 43.1754x_2 + 136.4352x_3 &= 29.4168\end{aligned}$$

18.3 Solve the following set of equations by (i) Matrix inversion method, and (ii) Gauss-Seidel method

$$\begin{aligned} 4x_1 + 12x_2 - 7x_3 + 6x_4 &= -1 \\ 3x_1 + 3x_2 - 2x_3 + x_4 &= 1 \\ 2x_1 + 9x_2 - 2x_3 + 11x_4 &= 8 \\ x_1 + 2x_2 + 3x_3 + 4x_4 &= 2 \end{aligned}$$

18.4 For the algorithms and sub-algorithms given in this chapter, write down the corresponding programs and subprograms respectively, in FORTRAN-IV.

for NOTES

Chapter Nineteen

Eigen Values and Eigen Vectors

19.1 SOLUTION TO HOMOGENEOUS SIMULTANEOUS LINEAR ALGEBRAIC EQUATIONS

As stated in Section 18.2.1 the Homogeneous Simultaneous Alegebraic equations will be of the form

$$\begin{aligned} a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 \ldots + a_{1,n} \cdot x_n &= 0 \\ a_{2,1} \cdot x_1 + a_{2,2} \cdot x_2 \ldots + a_{2,n} \cdot x_n &= 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots &= 0 \\ a_{n,1} \cdot x_1 + a_{n,2} \cdot x_2 \ldots + a_{n,n} \cdot x_n &= 0 \end{aligned}$$

Solution to a set of non-homogeneous simultaneous equations will yield a unique value to each variable.

But there may be more than one set of values that would satisfy the equations in the case of a set of homogeneous simultaneous equations. Such cases occur in problems related to the theory of elasticity, vibration analysis, electric-circuit-analysis etc. In case of analysis of stresses in a structural member, it may be necessary to find the maximum load or stress (largest Eigen Value) that the member can carry without failure. Similarly, in an electric circuit, it may be necessary to know the minimum current (smallest Eigen Value) that will damage the circuit.

To get the different possible sets of solutions the EIGEN VALUE technique has to be adopted.

19.2 EIGEN VALUE TECHNIQUE

In this technique the coefficient matrix is expressed as

$$\begin{bmatrix} a_{1,1}-\lambda) & a_{1,2} & a_{1,n} \\ a_{2,1} & (a_{2,2}-\lambda) & a_{2,n} \\ \text{etc} & & \\ a_{n,1} & a_{n,2} & (a_{n,m-\lambda}) \end{bmatrix}$$

The term λ in the above equations is a particular parameter of the system which has unknown values.

For example, in a vibrating system λ will be squares of natural frequency of the system.

Using the above coefficient matrix, the set of equations may be written as

$$\begin{bmatrix} (a_{1,1}-\lambda) & a_{1,2} & a_{1,n} \\ a_{2,1} & (a_{2,2}-\lambda) & a_{2,n} \\ \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & (a_{n,n}-\lambda) \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = 0$$

This can be compactly written as

$$(A-\lambda I)x = 0$$

where I = identify matrix.

The column matrix containing $x_1, x_2 \ldots\ldots x_n$ is called EIGEN VECTOR.

$x_1, x_2 \ldots\ldots x_n$ are known as COMPONENTS of EIGEN VECTOR.

The values obtained for λ are known as EIGEN VALUES or CHARACTERISTIC VALUES of Matrix A.

The above equation can also be written as

$$Ax = \lambda IX$$

To obtain the value of λ the characteristic equation of the matrix A is to be obtained. It is obtained by expanding the determinant A and it will be of the form·

$$(-1)_n\lambda_n + K_1\lambda^{n-1} + K_2\lambda^{n-2} \ldots\ldots + K_n = 0$$

The characteristic equation is also known as characteristic polynomial. Solving the above equation, the different values of λ are to be obtained. These are called Eigen Values. (There may be a maximum value and a minimum value in them). Substituting the requisite value of λ in the matrix A, the equation can be solved and the corresponding Eigen Vector (value of $x_1, x_2 \ldots\ldots x_n$) can be obtained.

EXAMPLE 19.1 Given the following homogeneous equations, find (1) Characteristic equation, (2) Eigen Values, and (3) Eigen Vector for each Eigen Value.

$$8x_1 - 6x_2 + 2x_3 = 0$$
$$-6x_1 + 7x_2 - 4x_3 = 0$$
$$2x_1 - 4x_2 + 3x_3 = 0$$

Solution:

$$A = \begin{bmatrix} 8 & -6 & 2 \\ -6 & 7 & -4 \\ 2 & -4 & 3 \end{bmatrix}$$

$$A - \lambda 1 I = \begin{bmatrix} 8-\lambda & -6 & 2 \\ -6 & 7-\lambda & -4 \\ 2 & -4 & 3-\lambda \end{bmatrix} = -\lambda^3 + 18x^2 - 45\lambda = 0 = \text{ Characteristic equation}$$

Solving the above cubic equation, the 3 values of λ that can be obtained are = 0, 3, 15 = EIGEN Values.

Substituting the above values in the matrix $A - \lambda I$, the corresponding EIGEN Vectors can be obtained.

(a) $\lambda = 0$:

$$(A - \lambda I)x = \begin{bmatrix} 8 & -6 & 2 \\ -6 & 7 & -4 \\ 2 & -4 & 3 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

Solving this, we will get

$$x_1 = 1$$
$$x_2 = 2$$
$$x_3 = 2$$

For Eigen Value $\lambda = 0$, the Eigen Vector is = $\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$.

(b) $\lambda = 3$:

$$(A - \lambda I)x = \begin{bmatrix} 8-3 & -6 & 2 \\ -6 & 7-3 & -4 \\ 2 & -4 & 3-3 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

Solving the above, the EIGEN VECTOR that will be obtained

$$\begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}$$

(c) $\lambda = 15$:

$$(A - \lambda I)x = \begin{bmatrix} 8-15 & -6 & 2 \\ -6 & 7-15 & -4 \\ 2 & -4 & 3-15 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

Solving the above, the EIGEN VECTOR that will be obtained

$$\begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix}$$

The above method of solution is called NON-TRIVIAL SOLUTION.

(In case CRAMER's rule is used, all the values will be 0. This is called a TRIVIAL solution)

In this case, the maximum or largest Eigen Value is 15 and the minimum 0. The minimum is also called 'lowest modulus'. The other values are called INTERMEDIATE EIGEN VALUES.

19.3 COMPUTER DETERMINATION OF EIGEN VALUES AND VECTORS

The method discussed above will not be suitable for computerisation since the order of the matrices that are come across may be very large and it will not be easy to form and solve the characteristic equation. So, any one of the following methods can be adopted: (1) POLYNOMIAL METHOD (FADDEEV-LEVERRIER method), and (2) ITERATION method (Power method). The ITERATION method is discussed here.

19.3.1 Iteration Method

The iteration method to find the largest Eigen Value (Eigen Value of the largest modulus) will be as follows:

1. Write the set of homogeneous simultaneous linear algebraic equations in the form

$$Ax = \lambda x$$

2. Assume some arbitrary values for the Eigen Vectors x_1, x_2 etc ... x_n, say 1. This vector is called 'TRIAL VECTOR'.
3. Equate $Ax = \lambda x$
4. Normalise the vector λx. (ie,) divide each element in λx by the highest modulus in the vector λx and make $\lambda =$ highest modulus. (This is also called 'normalisation factor')
5. With the new set of values of x, repeat steps 3 and 4.

This is continued till the components of Eigen Vector obtained from two successive iterations are same or differ by a very small value (EPSILON)

6. The normalisation factor obtained at the end will be the largest Eigen Value, say, λ_1

EXAMPLE 19.2 Find the largest Eigen Value of the Homogeneous simultaneous equations

$$x_1 + x_2 + 3x_3 = 0$$
$$x_1 - 2x_2 + x_3 = 0$$
$$3x_1 + x_2 + 3x_3 = 0.$$

Use iterative method.

Solution:

Step 1.

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 1 & -2 & 1 \\ 3 & 1 & 3 \end{bmatrix}$$

Step 2.

Assume Trial vector

$x_1 = 1, x_2 = 1, x_3 = 1$

$$Ax = \begin{bmatrix} 1 & 1 & 3 \\ 1 & -2 & 1 \\ 3 & 1 & 3 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 7 \end{bmatrix} = 7 * \begin{bmatrix} 0.71 \\ 0 \\ 1 \end{bmatrix} = \lambda * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

This was Iteration 1.

Step 3. (Perform next iteration with $x_1 = 0.71, x_2 = 0, x_3 = 1$)

$$Ax = \begin{bmatrix} 1 & 1 & 3 \\ 1 & -2 & 1 \\ 3 & 1 & 3 \end{bmatrix} * \begin{bmatrix} 0.71 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.71 \\ 2.71 \\ 5.13 \end{bmatrix} = 5.13 * \begin{bmatrix} 0.72 \\ 0.33 \\ 1 \end{bmatrix} = \lambda * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Step 4. Compare results in the last 2 iterations. Since there is no convergence, continue the iterations.

$$Ax = \begin{bmatrix} 1 & 1 & 3 \\ 1 & -2 & 1 \\ 3 & 1 & 3 \end{bmatrix} * \begin{bmatrix} 0.72 \\ 0.33 \\ 1 \end{bmatrix} = \begin{bmatrix} 4.06 \\ 1.04 \\ 5.49 \end{bmatrix} = 5.49 * \begin{bmatrix} 0.74 \\ 0.19 \\ 1 \end{bmatrix} = \lambda * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Step 5. Continue the iteration since there is no convergence.

$$Ax = \begin{bmatrix} 1 & 1 & 3 \\ 1 & -2 & 1 \\ 3 & 1 & 3 \end{bmatrix} * \begin{bmatrix} 0.74 \\ 0.19 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.95 \\ 1.352 \\ 5.41 \end{bmatrix} = 5.41 * \begin{bmatrix} 0.73 \\ 0.25 \\ 1 \end{bmatrix} = \lambda * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Step 6. Continue the iteration since there is no convergence.

$$Ax = \begin{bmatrix} 1 & 1 & 3 \\ 1 & -2 & 1 \\ 3 & 1 & 3 \end{bmatrix} * \begin{bmatrix} 0.73 \\ 0.25 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.97 \\ 1.25 \\ 5.44 \end{bmatrix} = 5.44 * \begin{bmatrix} 0.73 \\ 0.23 \\ 1 \end{bmatrix} = \lambda * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

The iteration is terminated since there is convergence in Values of x_1 and x_3. The largest Eigen Value = $\lambda_1 = 5.44$

$$\left.\begin{array}{l}\text{The corresponding} \\ \text{Eigen Vectors are}\end{array}\right\} \quad \begin{array}{l} x_1 = 0.73 \\ x_2 = 0.23 \\ x_3 = 1 \end{array}$$

19.3.2 Smallest Eigen Value

The smallest Eigen Value can be obtained by a similar iterative procedure.

For this, the matrix A^{-1}, which is the inverse of matrix A, is to be used. It is written as

$$A^{-1}x = \frac{1}{\lambda}x$$

EXAMPLE 19.3 Find the smallest Eigen Value of the homogeneous simultaneous equation by iterative method:

$$\begin{aligned} x_1 + 5x_2 + 2x_3 &= 0 \\ x_1 + x_2 + 7x_3 &= 0 \\ -3x_2 + 4x_3 &= 0 \end{aligned}$$

Solution:

Step 1.

$$A = \begin{bmatrix} 1 & 5 & 2 \\ 1 & 1 & 7 \\ 0 & -3 & 4 \end{bmatrix}$$

Step 2. Obtain A^{-1} (This part of the calculation is left to the student)

$$A^{-1} = \begin{bmatrix} -25 & 26 & -33 \\ 4 & -4 & 5 \\ 3 & -3 & 4 \end{bmatrix}$$

Step 3. Assume trial vector, $x_1 = 1, x_2 = 1, x_3 = 1$

$$A^{-1}x = \begin{bmatrix} -25 & 26 & -33 \\ 4 & -4 & 5 \\ 3 & -3 & 4 \end{bmatrix} * \begin{bmatrix} -32 \\ 5 \\ 4 \end{bmatrix} = -32 * \begin{bmatrix} 1 \\ -0.156 \\ -0.125 \end{bmatrix}$$

Step 4. Perform next iteration

$$\begin{bmatrix} -25 & 26 & -33 \\ 4 & -4 & 5 \\ 3 & -3 & 4 \end{bmatrix} * \begin{bmatrix} 1 \\ -0.156 \\ -0.125 \end{bmatrix} = \begin{bmatrix} -24.93 \\ 4 \\ 2.965 \end{bmatrix} = -24.93 * \begin{bmatrix} 1 \\ -0.16 \\ -0.1189 \end{bmatrix}$$

Step 5. Perform next iteration

$$\begin{bmatrix} -25 & 26 & -33 \\ 4 & -4 & 5 \\ 3 & -3 & 4 \end{bmatrix} * \begin{bmatrix} 1 \\ -0.16 \\ -0.1189 \end{bmatrix} = \begin{bmatrix} -25.236 \\ 4.05 \\ 3.02 \end{bmatrix} = -25.236 * \begin{bmatrix} 1 \\ -0.16 \\ -0.119 \end{bmatrix}$$

Since there is a convergence of values in the last 2 iterations, the process is terminated.

$$\begin{aligned} A^{-1}x &= \frac{1}{\lambda_2}x \text{ where } \lambda_2 = \text{ smallest Eigen Value} \\ \frac{1}{\lambda_2} &= -25.236 \\ \lambda_2 &= -0.0396 \end{aligned}$$

Corresponding to this Eigen Value the Eigen Vector :

$$\begin{aligned} x_1 &= 1 \\ x_2 &= -0.16 \\ \text{and} \quad x_3 &= -0.119 \end{aligned}$$

19.3.3 Intermediate Eigen Values

The intermediate Eigen Values may be computed using largest Eigen Value or the smallest Eigen Value. The calculation may be done by Gauss Algorithm or by Power (iterative) method. The iterative method is discussed here.

Let V= Eigen Vector

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix}$$

This can be expressed as a linear combination of the orthogonal Eigen Vectors (x_i)

(ie,)

$$V = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = C_1 * \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} + C_2 * \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} + \dots + C_n * \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

i.e.,

$$V = \sum_{i=1}^{n} C_i x_i$$

$$x = x_p - \sum_{i=1}^{r} C_i x_i$$

x_p = values of Eigen Vector used in trial vector or obtained in the previous iteration.

The Steps involved are:

(1) Calculate $G = ({x_1}^2 + {x_2}^2 + \dots\dots {x_n}^n)$ when $x_1, x_2, \dots\dots x_n$ are the Eigen Vectors corresponding to the largest or the smallest Eigen Value.

(2) Compute $C_i = \frac{1}{G} * X_r * X_T$

X_T = column matrix of trial vector

X_r = row matrix of Eigen Vector recently obtained

(3) Calculate a new trial column matrix X as

$$X = X_T - C_1 * X_r$$

(4) If largest Eigen Vector is used, multiply A with X_r to give better approximation of Eigen Vector

$$A * X_r = X$$

If smallest Eigen Value is used

$A^{-1} * X_r = X$ will give the better approximation.

(5) Normalise the vector obtained by dividing each element in it by the element of largest magnitude.

(6) Repeat the steps 2 to 5 until the value of Eigen Vector obtained in successive iterations differ by less than some predetermined very small value (Epsilon). Take the latest Eigen Vector and Eigen Value as the intermediate Eigen Vector and Value, respectively.

By performing the above steps on A and A^{-1}, two such sets of intermediate values can be obtainted. This technique is often called the 'SWEEPING OUT OF'' the last Eigen Vector before proceeding with next iteration.

≡ EXAMPLE 19.4 Using the largest Eigen Value obtained in the Example 19.1, find by iterative method an intermediate Eigen Value and the corresponding Eigen Vector.

Solution:

$$\left.\begin{array}{l}\text{Max value of Eigen Value} = 15\\ \text{Corresponding Eigen Vector} = (2,-2,1)\end{array}\right\}\text{From Example 19.1}$$

Step 1. $G = 2^2 + (-2)^2 + 1^2 = 9$

Step 2. ITERATION 1.

$$C_1 = \frac{1}{9}(2,-2,1) * \begin{bmatrix}1\\1\\1\end{bmatrix} = \frac{1}{9} = 0.1111$$

Step 3.

$$x = \begin{bmatrix}1\\1\\1\end{bmatrix} - 0.111 \begin{bmatrix}2\\-2\\1\end{bmatrix} = \begin{bmatrix}0.778\\1.222\\0.889\end{bmatrix}$$

Step 4.

$$\begin{bmatrix}8 & -6 & 2\\-6 & 7 & -4\\2 & -4 & 3\end{bmatrix} * \begin{bmatrix}0.778\\1.222\\0.889\end{bmatrix} = \begin{bmatrix}0.67\\0.33\\-.665\end{bmatrix}$$

Step 5.

$$\lambda(x) = 0.67 \times \begin{bmatrix}1\\0.5\\-0.992\end{bmatrix}$$

$$x = \begin{bmatrix}1\\0.5\\-0.992\end{bmatrix}$$

ITERATION 2.

Step 6.

$$C_2 = \frac{1}{9}[2,-2,1] * \begin{bmatrix}1\\0.5\\-0.992\end{bmatrix} = \frac{0.008}{9} = 0.00889$$

Step 7.

$$x = \begin{bmatrix}1\\0.5\\-0.992\end{bmatrix} - 0.00889 \times \begin{bmatrix}2\\-2\\1\end{bmatrix} = \begin{matrix}0.9822\\0.5178\\-1.0089\end{matrix}$$

Step 8.

$$\begin{bmatrix}8 & -6 & 2\\-6 & 7 & -4\\2 & -4 & 3\end{bmatrix} * \begin{bmatrix}0.9822\\0.5178\\-1.0089\end{bmatrix} = \begin{bmatrix}2.733\\1.767\\-3.133\end{bmatrix}$$

Step 9.

$$\lambda x = -3.133 \times \begin{bmatrix} -0.872 \\ -0.583 \\ 1 \end{bmatrix}$$

ITERATION 3.

Step 10.

$$C_3 = \frac{1}{9}(2, -2, 1) \times \begin{bmatrix} -0.872 \\ -0.583 \\ 1 \end{bmatrix} = \frac{0.422}{9} = 0.047$$

Step 11.

$$x = \begin{bmatrix} -0.872 \\ -0.583 \\ 1 \end{bmatrix} - 0.047 \times \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix} = \begin{matrix} -0.966 \\ -0.489 \\ 0.953 \end{matrix}$$

Step 12.

$$\begin{bmatrix} 8 & -6 & 2 \\ -6 & 7 & -4 \\ 2 & -4 & 3 \end{bmatrix} * \begin{bmatrix} -0.966 \\ -0.489 \\ 0.953 \end{bmatrix} = \begin{bmatrix} -2.888 \\ -1.439 \\ 2.883 \end{bmatrix}$$

Step 13.

$$\lambda x = 2.88 \times \begin{bmatrix} -1 \\ -0.5 \\ 1 \end{bmatrix}$$

The iterations may be continued or may be terminated. It will be found one of the intermediate Eigen Value ≈ 2.88 and the corresponding Eigen Vectors are

$$x_1 = -1$$
$$x_2 = -0.5$$
$$\text{and} \quad x_3 = 1$$

The above are intermediate values calculated using largest Eigen Value and matrix A.

Similarly, using the smallest Eigen Value and the inverse matrix A^{-1}, another set of intermediate values can be computed. This is left as an exercise to the student.

Summary

The homogeneous simultaneous linear algebraic equations will have more than one set of solutions. If CRAMER's Rule is applied to them, all the values obtained will be zero. This is called a TRIVIAL SOLUTION.

To get other (non-zero) solutions which are called NON-TRIVIAL SOLUTIONS, Eigen Vector technique is to be adopted.

In case of computerisation, an iterative method called POWER METHOD is used to find the largest and the smallest Eigen Values as also the corresponding Eigen Vectors. From these, the intermediate Eigen Vector and Eigen Values can also be obtained by iterative procedure.

Exercise/Questions

19.1 Differentiate between HOMOGENEOUS and NON-HOMOGENEOUS SIMULTANEOUS EQUATIONS.

19.2 Define—(1) EIGEN VALUE, (2) EIGEN VECTOR.

19.3 Find the EIGEN VALUES and the corresponding EIGEN VECTORS for

$$\begin{aligned} 4x_1 - 5x_2 + 6x_3 &= 0 \\ 3x_1 + 4x_2 - 5x_3 &= 0 \\ 2x_1 + x_2 + 3x_3 &= 0 \end{aligned}$$

19.4 Find the largest Eigen Value for the equations in the above problem and the corresponding Eigen Vector, by iterative method. Find also an INTERMEDIATE EIGEN VALUE and the corresponding EIGEN VECTOR.

19.5 Find the smallest Eigen Value for the equations in the Problem (19.3) and the corresponding Eigen Vector by iterative method. Find also an intermediate Eigen Value and corresponding Eigen Vector.

for NOTES

Chapter Twenty

Numerical Integration

20.1 INTRODUCTION

In Engineering applications, problems of numerical integrations arise. In such cases it will be required to compute the value of a definite integral, say, in the form—

$\int_a^b f(x)dx$, where $f(x)$ is not known explicitly.

Numerical integration is also called NUMERICAL QUADRATURE. It essentially consists of finding a close approximation to the area under a curve of a function $f(x)$. (refer Fig 20.1)

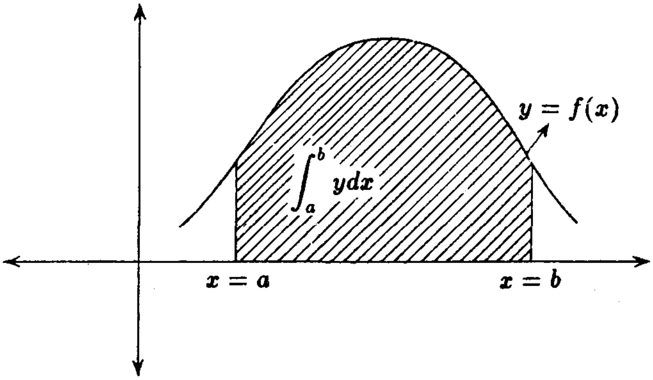


Fig 20.1

The numerical techniques formulated for this purpose are: (1) The RECTANGLE RULE, (2) THE TRAPEZOID RULE, and (3) SIMPSON'S RULE.

Of these, the RECTANGLE rule is very inaccurate and hence not discussed here.

20.2 THE TRAPEZOID RULE

If a function $f(x)$, ie, $y = f(x)$ is to be integrated between limits $x = a$ and $x = b$, a close approximation to the area under the curve representing $f(x)$ is obtained by dividing the interval a to b into n narrow strips each of width, say, $h(\delta x)$, computing the area of each strip, treating it as a trapezoid (trapezeum) and adding the areas of all the strips. (refer Fig 20.2)

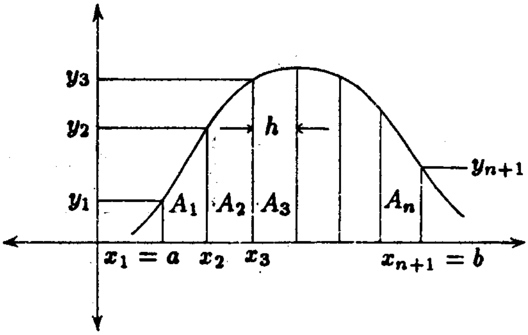


Fig 20.2

If there are n strips, there will be $n + 1$ values of x, say, $x_1, x_2 \ldots\ldots x_{n+1}$ and $x_1 = a$ and $x_{n+1} = b$.

Corresponding to x_1 to x_{n+1} there will be $(n + 1)$ values of y, namely, $y_1, y_2, \ldots\ldots y_{n+1}$ which are given by $y_i = f(x_i)$

If the areas of n strips are denoted as $A_1, A_2, A_3 \ldots$ etc A_n,

$$A_1 = h\left(\frac{y_1 + y_2}{2}\right); \quad A_2 = h\left(\frac{y_2 + y_3}{2}\right); \quad A_3 = h\left(\frac{y_3 + y_4}{2}\right); \quad \text{etc}$$

$$A_{n-1} = h\left(\frac{y_{n-1} + y_n}{2}\right); \quad \text{and} \quad A_n = h\left(\frac{y_n + y_{n+1}}{2}\right)$$

The value of the definite integral $= A$

$$= A_1 + A_2 + \ldots\ldots\ldots A_{n-1} + A_n = \frac{h}{2}(y_1 + 2y_2 + 2y_3 + \ldots\ldots + 2y_n + y_{n+1})$$

$$= \frac{h}{2}\left(y_1 + 2\sum_{i=2}^{n} y_i + y_{n-1}\right)$$

EXAMPLE 20.1 Evaluate $A = \int_1^2 (x^2 + 2x + 3)dx$ by the use of Trapezoid rule.

Solution:

Step 1.

Divide the interval $x = 1$ to $x = 2$ into 10 strips of width 0.1 $\delta x = h = 0.1$

Step 2.

The values of x and the corresponding values of y within the above interval are as follows

$$\begin{array}{ll}
x_1 = 1 & y_1 = 1.0^2 + 2 \times 1.0 + 3 = 6.00 \\
x_2 = 1.1 & y_2 = 1.1^2 + 2 \times 1.1 + 3 = 6.41 \\
x_3 = 1.2 & y_3 = 1.2^2 + 2 \times 1.2 + 3 = 6.84 \\
x_4 = 1.3 & y_4 = 1.3^2 + 2 \times 1.3 + 3 = 7.29 \\
x_5 = 1.4 & y_5 = 1.4^2 + 2 \times 1.4 + 3 = 7.76 \\
x_6 = 1.5 & y_6 = 1.5^2 + 2 \times 1.5 + 3 = 8.25 \\
x_7 = 1.6 & y_7 = 1.6^2 + 2 \times 1.6 + 3 = 8.76 \\
x_8 = 1.7 & y_8 = 1.7^2 + 2 \times 1.7 + 3 = 9.29 \\
x_9 = 1.8 & y_9 = 1.8^2 + 2 \times 1.8 + 3 = 9.84 \\
x_{10} = 1.9 & y_{10} = 1.9^2 + 2 \times 1.9 + 3 = 10.41 \\
x_{11} = 2 & y_{11} = 2^2 + 2 \times 2 + 3 = 11.00
\end{array}$$

Step 3.

$$\begin{aligned}
A &= \frac{h}{2}[y_1 + 2(y_2 + y_3 + \ldots\ldots + y_{10}) + y_{11}] \\
&= \frac{0.1}{2}[6.0 + 2(6.41 + 6.84 + 7.29 + 7.76 + 8.25 + 8.76 + 9.29 + 9.84 + 10.41) + 11.00] \\
&= \frac{00.1}{2}[6 + 149.7 + 11] = 8.335
\end{aligned}$$

$$\int_1^2 (x^2 + 2x + 3)dx = 8.335.$$

The accuracy of the result will be more if the interval is divided into strips of lesser width, say .01 or .001.

In the computer algorithm the entire interval a to b is taken as a single strip in the beginning ($h = b - a$) and then the area A is found. Then the value of h is halved in successive iterations until the absolute value of difference in values of A in two successive iterations is less than a predetermined small value ϵ (Epsilon).

20.2.1 Algorithm: Trapezoid

(Given $y = f(x)$ and the limits a and b as also ϵ (Epsilon) the desired accuracy, to compute $A = \int_a^b ydx$)

```
1. (INITIALISE),
   A ← 0
   h ← b − a
2. AOLD ← A
3. (Calculate the number of strips (sub intervals))
   n ← (b − a)/h
   k ← n + 1
4. (Compute the ordinates $y_1$ to $y_k$)
   Repeat for I = 1, 2, 3 ...... k
   x(I) = a + h(I − 1)
   y(I) = f(x(I))
```

```
5. SUM ← 0.0
6. Repeat for J = 2, 3, 4, ...... n
   SUM ← SUM +y(J)
7. (Evaluate the definite integral,
   A ← $\frac{h}{2}$ * (y(1) + 2* (SUM) + y(n + 1))
8. (Compare results of 2 successive iterations)
   IF (ABS(A—AOLD)< E) then GOTO 9
9. ELSE
   h ← h/2
   GOTO 2
10. (OUTPUT THE RESULT)
    WRITE ('THE VALUE OF THE DEFINITE INTEGRAL =', A)
11. (FINISHED)
    EXIT.
```

Though the trapezoid rule is not as inaccurate as the rectangle rule, still it is far from accurate. The accuracy in the result may be improved by ROMBERG integration which is also called 'RICHARDSON's deferred approach to the limit'. This is left to the student for future study.

20.3 SIMPSON'S RULE

The evaluation of a definite integral by SIMPSON'S RULE consists of connecting successive groups of 3 points on the curve (belonging to 2 adjacent strips) by a second degree curve (parabola) and summing the areas under these parabolas to obtain the approximate overall area under the curve. As such, the number of strips into which the interval $x = a$ to $x = b$ is divided should always be even, and hence the total number of ordinates (y) should be odd (since no. of ordinates = no. of strips + 1).

Taking 2 adjacent strips each of width $h = \delta x$, with their 3 points y_1, y_2, y_3 lying on a parabolic curve ($y = ax^2 + bx + c$), the area within the 2 strips can be computed as follows (refer Fig 20.3)

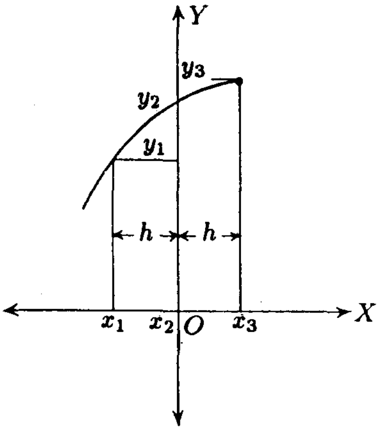


Fig 20.3

Choosing the 2 strips on either side of Y axis, the 3 points on the curve are $(x_1, y_1), (x_2, y_2)$, and (x_3, y_3).

Since, x_2 is on the origin itself, $x_2 = 0$ and $x_1 = x_2 - h = -h$, and $x_3 = x_2 + h = h$

Since $y = ax^2 + bx + c$

$$\begin{aligned} y_1 &= a(-h)^2 + b(-h) + c = ah^2 - bh + c \\ y_2 &= a(0)^2 + b(0) + c = c \\ y_3 &= ah^2 + bh + c \end{aligned}$$

$$\int_{-h}^{h} ax^2 + bx + c = \left[\frac{ax^3}{3} + \frac{bx^2}{2} + cx\right]_{-h}^{h} = \frac{2}{3}ah^3 + 2ch = \text{Area of 2 strips } = S_1$$

The values of a and c can be evaluated from the above equations of y_1, y_2 and y_3

$$c = y_2; \quad a = \frac{y_1 - 2y_2 + y_3}{2h^2}.$$

Substituting this in the area for 2 strips

$$S_1 = \frac{h}{3}(y_1 + 4y_2 + y_3)$$

If the area under the curve is divided into n strips, there will be $n/2$ such areas, say, $S_1, S_2, \ldots S_{n/2}$. (refer Fig 20.4).

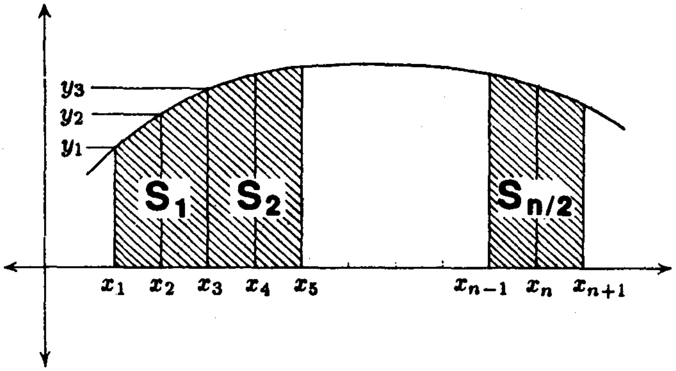


Fig 20.4

It is proved $S_1 = \frac{h}{3}(y_1 + 4y_2 + y_3)$

Similarly, $S_2 = \frac{h}{3}(y_2 + 4y_4 + y_5)$

$S_3 = \frac{h}{3}(y_5 + 4y_6 + y_7)$ etc

and $S_{n/2} = \frac{h}{3}(y_{n-1} + 4y_n + y_{n+1})$

Summing the above

$$\begin{aligned} A &= S_1 + S_2 + S_3 \ldots\ldots S_{n/2} \\ &= \frac{h}{3}(y_1 + 4y_2 + 2y_3 + 4y_4 + 2y_5 + \ldots\ldots 2y_{n-1} + 4y_n + y_{n+1}) \end{aligned}$$

EXAMPLE 20.2 Evaluate the definite integral given in Example 20.1 by the use of Simpson's Rule.

Solution:

(Steps 1 and 2 are same as in Problem 20.1)

Step 3.

$$\begin{aligned}A &= \frac{h}{3}[y_1 + 4(y_2 + y_4 + y_6 + y_8 + y_{10}) + 2(y_3 + y_5 + y_7 + y_9) + y_{11}]\\ &= \frac{0.1}{3}[6.00 + 4(6.41 + 7.29 + 8.25 + 9.29 + 10.41) + 2(6.84 + 7.76 + 8.76 + 9.84) + 11.00]\\ &= 0.1[6 + 4(41.65) + 2(33.20) + 11] = 8.333\end{aligned}$$

$$\int_1^2 x^2 + 2x + 3 = 8.333.$$

In this case also, the accuracy will be more if the value of h is small.

In the computer algorithm for Simpson's Rule, the interval a to b is initially divided into 2 strips each of width $h = (b - a)/2$ and then the area A is found. Then the value of h is halved in successive iterations until the absolute value of the difference in values of A in 2 successive iterations is less than a predetermined small value ϵ (Epsilon).

20.3.1 Algorithm : Simpson

Given $y = f(x)$ and the limits a and b as also ϵ (Epsilon) the desired accuracy, to compute $A = \int_a^b y dx$.

```
1. (INITIALISE)
   A ← 0
   h ← (b − a)/2
2. AOLD ← A
3. Calculate the number of strips (subintervals)
   n = (b − a)h
4. (Compute sum)
   SUM ← f(a) + 4.0 * f(a + h) + f(b)
   Repeat for I = 2, 4, 6, ......n − 2
   SUM ← SUM +2 * f(a + I * h) + 4 * f(a + (I + 1) * h)
5. (Evaluate the definite integral)
   A ← $\frac{h}{3}$* SUM
6. (Test for convergence)
   IF (ABS (A-AOLD)< $\epsilon$) then GOTO 8
7. ELSE
   h ← h/2
   Go To 2
8. (OUTPUT THE RESULT)
   WRITE ('THE VALUE OF THE DEFINITE INTEGRAL =', A)
9. (FINISHED)
   EXIT.
```

The Simpson's Rule method is found to give more accurate result than the Trapezoid Rule.

Summary

The digital computers, cannot perform analytical operations like absolute integration. But it can perform numerical integration, ie, integration within numerical limits, ie, evaluation of numerical integrals, using certain numerical methods.

The 2 numerical methods used are (1) Trapezoid Rule, and (2) Simpson's Rule. Of these, the Simpson's Rule is more accurate.

The Trapezoid Rule approximates the area under a curve by summing the areas of trapezoids, of uniform width, formed by connecting successive points on the curve by straight lines.

Simpson's Rule consists of connecting successive groups of three points on the curve by parabola and summing the areas under the parabolas to obtain the approximate area under the curve.

Exercise/Questions

20.1 Explain why integration as such cannot be performed in a digital computer.

20.2 Evaluate $\int_{\pi_2}^{\pi} \cos^2 x dx$ using Trapezoid Rule with $h = 0.001$

20.3 Evaluate $\int_1^2 x^2 c^x dx$ using the trapezoid algorithm given in the text.

20.4 Evaluate $\int_1^{\pi/2} (1 - \sin x) dx$ using 50 strips of equal width by Simpson's Rule.

20.5 Evaluate $\int_{-1}^{1} (x^3 - 4x^2 + 3x - 5) dx$ using the ALGORITHM SIMPSON in the text.

20.6 Write down FORTRAN programs for (1) Numerical integration using TRAPEZOIDAL RULE, and (2) Numerical integration by SIMPSON'S RULE.

for NOTES

Chapter Twentyone

Numerical Solution to Ordinary Differential Equations

21.1 INTRODUCTION

In the solution of Engineering problems it may be often necessary to evaluate differential equations. So, special numerical methods have been evolved for this purpose since the digital computer cannot, as such, solve them.

Problems involving differential equations are classified into two types for purpose of formulating the numerical methods of solution—(1) INITIAL-VALUE problems (ordinary differential equations), and (2) BOUNDARY-VALUE problems.

Problems in which the condition to be satisfied by the solution or its derivatives is given at the point where the integration is to begin, are called Initial value problems.

eg,

$$\frac{dy}{dx} = \sqrt{1 - y^2}$$

when $x = 0, y = 0$

(This is also written as $y(0) = 0$)

Problems in which conditions are specified at both ends of the interval (initial value and fina value) over which the integration is to be performed, are called Boundary-value problems. Th INITIAL value problems alone are discussed in this chapter.

21.2 INITIAL VALUE PROBLEMS

The INITIAL VALUE problem involves: given a differential equation with a known initial value ($y(x_0) = y_0$) to find the subsequent values of y for small increments (dx) to x.

Some of the numerical methods for solving INITIAL VALUE problems are: (1) PICARD's method, (2) EULER's method, (3) MODIFIED EULER's method, (4) RUNGE-KUTTA's SECOND ORDER method, (5) RUNGE-KUTTA's FOURTH ORDER method, (6) TAYLOR SERIES method, and (7) PREDICTOR-CORRECTOR method.

21.3 PICARD'S METHOD

Consider the differential equation

$$\frac{dy}{dx} = f(x, y)$$

with an initial value y_0 when $x = x_0$, it is required to find values of y for small increments in x.

Integrating the above equation,

$$y = y_0 + \int_{x_0}^{x} f(x, y)dx$$

y in the function $f(x, y)$ is called an INTEGRAL EQUATION. Such an equation can be solved by the method of successive approximation.

The first approximation will be :

$y_1 = y_0 + \int_{x_0}^{x} f(x, y_0)dx$.

Using y_0 the above equation can be solved and value of y_1 obtained.

Using y_1, y_2 (the value of y for the next increment in x) can be determined as

$$y_2 = y_0 + \int_{x_0}^{x} f(x, y_1)dx$$

Proceeding as above, the values of y_3, y_4 etc ... y_n can be obtained.

$$y_n = y_0 \int_{x_0}^{x} f(x, y_{n-1})dx$$

EXAMPLE 21.1 Solve the differential equation $\frac{dy}{dx} = \frac{x^2}{y^2+1}$ given the initial condition that $y(0) = 0$ (ie, when $x = 0, y = 0$) to find values of y corresponding to $x = 0.1, 0.2$ and 0.3.

Solution:

$$y = \int_0^x \frac{x^2}{y^2+1}dx$$

$$y_1 = y_0 + \int_{x_0}^{x} f(x, y_0)dx = 0 + \int_{x_0}^{x} \frac{x^2}{0+1}dx = \int_{x_0}^{x} x^2 dx = \frac{1}{3}x^3$$

$$y_2 = y_0 + \int_{x_0}^{x} f(x, y_1)dx = 0 + \int_{x_0}^{x} \frac{x^2}{\frac{x^6}{9}+1}dx \approx \frac{1}{3}x^2$$

Since, $y_2 \approx y_1$ the process may terminate here

$$x = 0.1 \quad y = \frac{1}{3} \times 0.1^2 = 0.0033$$
$$x = 0.2 \quad y = \frac{1}{3} \times 0.2^2 = 0.0133$$
$$x = 0.3 \quad y = \frac{1}{3} \times 0.3^2 = 0.03$$

x	y
0	0
0.1	0.0033
0.2	0.0133
0.3	0.03

The defect in this method is more and more difficult integrations may have to be performed as we proceed for higher and higher approximations. Also, the result is often in the form of a series.

21.4 EULER'S METHOD

This is sometimes called EULER-CAUCHY method. This gives the solution of a differential equation in the form of a set of tabulated values.

If the differential equation is

$\frac{dy}{dx} = f(x, y)$ with the initial condition that $y = y_0$ when $x = x_0$, the value of y_1 corresponding to $x_1 = x + h$, where h is a small increment in x can be determined as follows

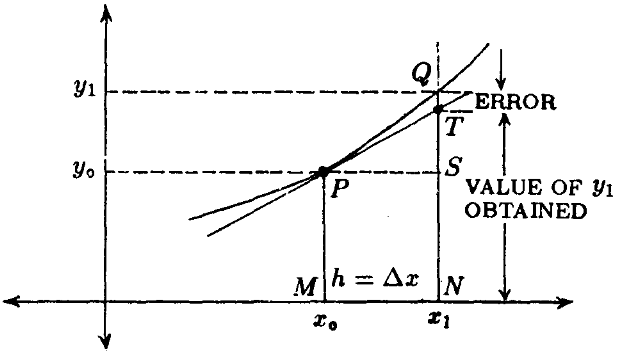


Fig 21.1

Referring to Fig 21.1, drawing the tangent PT through point $p(x_0, y_0)$, the value of

$$\begin{aligned} y_1 &= NQ \approx NT = NS + ST \\ &= y_0 + PS \tan Q \\ &= y_0 + h \tan Q \end{aligned}$$

Since $\tan Q = \frac{dy}{dx} = f(x, y)$, it can be written as

$$y_1 = y_0 + hf(x_0, y_0).$$

Of course this will have an error represented by TQ in the Fig 21.1. This error is called LOCAL TRUNCATION ERROR. To minimise the error, h can be taken as small a value as possible and the value of y at the given value of x can be obtained by successive approximations.

$$\begin{aligned} y_1 &= y_0 + hf(x_0, y_0) \\ y_2 &= y_1 + hf(x_1, y_1) \\ y_3 &= y_2 + hf(x_2, y_2) \text{ etc} \\ \text{and} \quad y_n &= y_{n-1} + hf(x_{n-1}, y_{n-1}) \end{aligned}$$

EXAMPLE 21.2 Given that $y' = x\sqrt{y}$ and that $y = 1$ when $x = 1$, find value of y when $x = 1.03$

Solution:

Given $\frac{dy}{dx} = xy^{1/2}$ and $y(1) = 1$, to find $y(1.03)$.

This can be found by successive calculations taking $h = \delta x = 0.01$

$$\begin{aligned} x_1 &= 1.01 \\ y_1 &= y_0 + hf(x_0, y_0) = 1 + 0.01 * (1 * 1^{0.5}) = 1.01 \\ x_2 &= 1.02 \\ y_2 &= y_1 + hf(x_1, y_1) = 1.01 + 0.01 * (1.01 * 1.01^{0.5}) = 1.0202 \\ x_3 &= 1.03 \\ y_3 &= y_2 + hf(x_2, y_2) = 1.0202 + 0.01 * (1.02 * 1.0202^{0.5}) = 1.0305 \\ y(1.03) &= 1.0305 \end{aligned}$$

21.5 MODIFIED EULER'S METHOD

To minimise the LOCAL TRUNCATION ERROR, that is inevitable in the EULER's method, a modified Euler's method is often used. This method is known also by other names (1) EULER-TRAPEZOIDAL method, (2) EULER PREDICTOR-CORRECTOR method, (3) The IMPROVED EULER method, and (4) HEUN's method. (Some authors call this EULER-CAUCHY method).

The error in Euler's method is due to the assumption that the slope of the tangent at a point of the curve is equal to the chord connecting it and the next point..

This error is minimised by this modified EULER's method which is of PREDICTOR-CORRECTOR type.

The PREDICTOR step comprises getting, the approximate value of y_1 as in Euler's method. The CORRECTOR step comprises of modifying or refining it. As derived under Section 21.4, the Predictor value of y_1, say

$$py_1 = y_0 + h\frac{dy}{dx} = y_0 + hf(x_0, y_0)$$

Let s_0 be the slope at point x_0, y_0

Let s_1, be the slope at point py_1 on the curve. (refer Fig 21.2)

If s is the slope at the point y_1, which is more or less accurate value, it is taken as $(s_1 + s_2)/2$ ie, $\frac{dy_1}{dx_1} = \frac{S_1+S_2}{2} = \left(\frac{dy_0}{dx_0} + \frac{dpy_1}{dx_1}\right)/2$

So CORRECTOR value of y_1,

$$C_1y_1 = y_0 + h\frac{dy_1}{dx_1} = y_0 + \frac{h}{2}\left(\frac{dy_0}{dx_0} + \frac{dpy_1}{dx_1}\right) = y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, py_1))$$

The computation of corrector values can be made iterative as follows

$$C_2y_1 = y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, C_1y_1));$$
$$C_3y_2 = y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, C_2y_1)) \text{ etc}$$

The iterations may be continued until, say, $C_iy_1 \approx C_{i+1}y_1$.

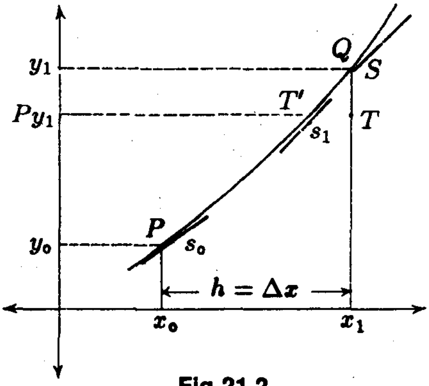


Fig 21.2

EXAMPLE 21.3 Given that $y' = x + y$ and given that $y(0) = 1$, find value of y when $x = 0.1$.

Solution:

Assuming $h = 0.05$

Step 1.

$x_1 = x + h = 0 + 0.05 = 0.05$

To find y_1 :

ITERATION 1.

$$Py_1 = y_0 + hf(x_0, y_0) = y_0 + 0.05(x_0 + y_0) = 1 + 0.05(0 + 1) = 1.05$$
$$C_1y_1 = y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, py_1)) = 1 + \frac{0.05}{2}((0 + 1) + (0.05 + 1.05)) = 1.0525$$

ITERATION 2.

$$C_2y_1 = y_0 + \frac{h}{2}((f(x_0, y_0) + f(x_1, C_1y_1)) = 1 + \frac{0.05}{2}((0 + 1) + (0.05 + 1.0525)) = 1.052563.$$

ITERATION 3.

$$C_3y_1 = y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, C_2y_1) = 1 + \frac{0.05}{2}((0 + 1) + (0.05 + 1.05263)) = 1.0525657.$$

$\because \quad C_2y_1 \approx C_3y_1$

y_1 may be taken as $= 1.0525657 = \mathbf{1.0526}$.

STEP 2. $x_2 = x_1 + h = 0.05 + .05 = .1$

(To find y_2)

ITERATION 1.

$$Py_2 = y_1 + hf(x_1, y_1) = 1.0526 + .05(.05 + 1.0526) = 1.10513$$

$$C_1y_2 = y_1 + \frac{h}{2}((f(x_1, y_1) + f(x_2, py_2))$$

$$= 1.0526 + \frac{0.05}{2}((0.05 + 1.0526) + (0.1 + 1.10513)) = 1.1103$$

ITERATION 2.

$$C_2y_2 = y_1 + \frac{h}{2}(f(x_1, y_1) + f(x_1, C_1y_2)) = 1.0526 + \frac{0.05}{2}((0.05 + 1.0526) + (0.1 + 1.1103)) = 1.1104$$

Since, $C_1y_2 \approx C_2y_2$

y_2 can be taken as $= 1.1104$

$y(0.1) = 1.1104$

ie, when $x = 0.1, y = 1.1104$

21.6 SECOND ORDER RUNGE-KUTTA METHOD

This is also known as RUNGE method or POLYGON method. This method is also known as SINGLE STEP method since it requires the knowledge of only y_i to compute y_{i+1}.

Using the same notations as in the previous methods, this method involves the following steps:

$$k_1 = h(f(x_0, y_0))$$

$$k_2 = h\left(f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right)\right)$$

$$y_1 = y_0 + k_2$$

This can be geometrically derived (refer Fig 21.3)

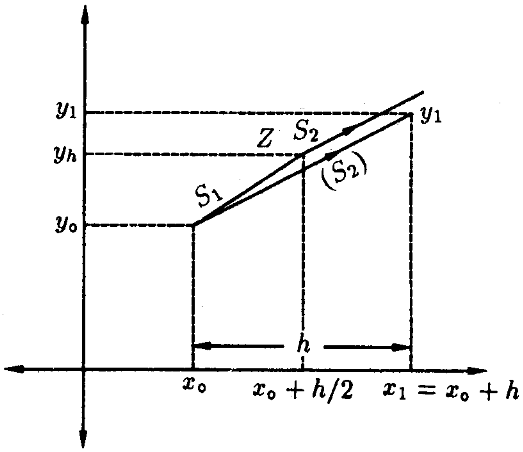


Fig 21.3

Slope $S_1 = \frac{dy_0}{dx_0} = f(x_0, y_0)$

When x is incremented from x_0 to $(x_0 + h/2)$, let y_h be the corresponding value of y.

$$f\left(x_0 + \frac{h}{2}, y_h\right) \quad \text{will be} \quad \frac{dy_h}{d(x_0 + h/2)}$$

which will be slope S_2 at point Z. Draw a line passing through (x_0, y_0) and parallel to S_2. If this cuts the vertical through x_1, the intercept is taken to give the value of y_1.

Therefore, $y_1 = y_0 + hf(x_1 + \frac{h}{2}, y_h)$

Referring to the Fig 21.3.

Slope $S_1 = \frac{y_h - y_0}{h/2}$

So, $y_h = y_0 + S_1\frac{h}{2}$

Also $S_1 = \frac{dy_0}{dx_0} = f(x_0, y_0)$

Putting $K_1 = hf(x_0, y_0)$

$$S_1\frac{h}{2} = \frac{h}{2}f(x_0, y_0) = \frac{k_1}{2}$$

$$y_h = y_0 + S_1\frac{h}{2} = y_0 + \frac{k_1}{2}$$

$$y_1 = y_0 + hf\left(x_1 + \frac{h}{2}, y_h\right) = y_0 + hf(x_1 + \frac{h}{2}, y_0 + \frac{k_1}{2}\Big) = y_0 + k_2$$

where $k_2 = hf(x_1 + \frac{h}{2}, y_0 + \frac{k_1}{2})$

EXAMPLE 21.4 Given $y' = x - y$, and $y(0) = 1$, find $y(0.1)$ and $y(0.2)$.

Solution:

(a) $x_0 = 0, y_0 = 1$, and $h = 0.1$

$$k_1 = hf(x_0, y_0) = 0.1(x_0 - y_0) = 0.1(0 - 1) = -0.10$$

$$k_2 = hf\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right) = 0.1 * f\left(0 + \frac{0.1}{2}, 1 + \frac{1 - 0.10}{2}\right)$$

$$= 0.1 * f(0.05, 0.95) = 0.1(0.05 - 0.95) = -0.09$$

$$y_1 = y_0 + k_2 = 1 - 0.09 = 0.91 = y(0.1)$$

(b) $y(0.2)$ can be found using $y(0.1)$

$$k_1 = hf(x_1, y_1) = 0.1 * f(0.1, 0.91) = 0.1 * (0.1 - 0.91) = -0.081$$

$$k_2 = hf\left(x_1 + \frac{h}{2}, y_1 + \frac{k_1}{2}\right) = 0.1 * f(0.1 + 0.05, 0.91 + (-0.0405))$$

$$= 0.1 * f(0.105, 0.90595) = 0.1(0.105 - 0.90595) = -0.080095$$

$$y_2 = y_1 + k_2 = 0.91 - 0.080095 \approx 0.83$$

$$y(0.2) = 0.83$$

21.7 RUNGE-KUTTA FOURTH ORDER METHOD

A few higher order RUNGE-KUTTA methods are available for numerical solution to differential equations. One that is used in computer applications is the set of fourth order formulas, which are:

$$\Delta y = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = hf(x_0, y_0); \quad k_2 = hf\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right)$$
$$k_3 = hf\left(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}\right); \quad k_4 = hf(x_0 + h, y_0 + k_3)$$
$$x_1 = x_0 + h, \quad \text{and} \quad y_1 = y_0 + \Delta y$$

The proof of the above formulae is beyond the scope of this book.

EXAMPLE 21.5 Given $y' = y - x$, where $y(0) = 2$, find $y(0.1)$ and $y(0.2)$ using RUNGE-KUTTA's fourth order method.

Solution:

$$x_0 = 0; \quad y_0 = 2; \quad h = 0.1$$
$$f(x, y) = y - x$$
$$k_1 = hf(x_0, y_0) = hf(0, 2) = h(2 - 0) = 0.2$$
$$k_2 = hf\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right) = hf(0 + 0.05, 2 + 0.1) = 0.1(2.1 - 0.05) = 0.205$$
$$k_3 = hf\left(x + \frac{h}{2}, y_0 + \frac{k_2}{2}\right) = 0.1(f(0.05, 2 + 0.1025) = 0.20525$$
$$k_4 = hf(x_0 + h, y_0 + k_3) = 0.1f(0.05, 2 + 0.20525) = 0.210525$$
$$\Delta y = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = \frac{1}{6}(0.2 + 2 \times 0.205 + 2 \times 0.20525 + 0.210525) = 0.20517$$
$$y_1 = y_0 + \Delta y = 2 + 0.20517 = 2.20517, \text{ ie, } y(0.1) = 2.20517$$

(using $y(0.1)$, the value of $y(0.2)$ can be obtained. This is left to the student as an exercise).

21.8 TAYLOR'S SERIES METHOD

Let the given differential equation be

$$y' = \frac{dy}{dx} = f(x, y)$$

Assuming when $x = x_0$, the value of y, say, y_0 is known, the value of y, say, y_1, when $x = x_1 = x_0 + h$ can be obtained by TAYLOR's series.

$$y_1 = y_0 + hy_0' + \frac{h^2 y_0''}{2!} + \frac{h^3 y_0'''}{3!} + \frac{h^4 y_0''''}{4!} \text{ etc}$$

The truncation of the summing up of the terms is done depending on the desired accuracy.

EXAMPLE 21.6 Given $y' = x - y^2$ and $y(0) = 1$ find $y(0.1)$ by Taylor's series method.

Solution:

$$
\begin{aligned}
y' &= \frac{dy}{dx} = x - y^2 \\
x_0 &= 0; y_0 = 1, h = 0.1 \\
x &= x_0 + h = 0.1 \\
y'_0 &= 0 - 1^2 = -1 \\
y'' &= \frac{d^2y}{dx^2} = 1 - 2yy' \\
y''_0 &= 1 - 2 \times 1 \times -1 = 3 \\
y''' &= \frac{d^3y}{dx^3} = -2yy'' - 2y'^2 \\
y'''_0 &= -2 \times 1 \times 3 - 2(-1)^2 = -8 \\
y'''' &= \frac{d^4y}{dx^4} = -2yy''' - 6y'y'' \\
y''''_0 &= -2 \times 1 \times (-8) - 6 \times (-1) \times 3 = 34 \\
y_1 &= y_0 + hy'_0 + \frac{h^2y''_0}{2!} + \frac{h^3y'''_0}{3!} + \frac{h^4y''''_0}{4!} \text{ etc} \\
&= 1 + 0.1(-1) + \frac{0.1^2(3)}{1 \times 2} + \frac{0.1^3(-8)}{1 \times 2 \times 3} + \frac{0.1^4(34)}{1 \times 2 \times \ 3 \times 4} + \cdots\cdots \\
&= 1 - 0.1 + 0.015 - 0.00133 + 0.00014 + \ldots \\
&= 0.9138 \\
y_1 &= y(0.1) = 0.9138
\end{aligned}
$$

Taking $y(0.1) = 0.9138, y(0.2)$ can be obtained by similar method.

21.9 PREDICTOR-CORRECTOR METHODS

The PREDICTOR-CORRECTOR methods involve the use of one formula, to first get an approximate value of the next y_i (PREDICTOR FORMULA) and then apply more accurate formula (CORRECTOR FORMULA), to make successive improvements in the value y_i. The modified EULER's method is also of this type.

The other PREDICTOR-CORRECTOR methods are (1) MILNE's method, (2) HAMMING's method, and (3) ADAM-SMOULTON method. Of these the MILNE's method alone is discussed here.

21.9.1 MILNE's (Predictor-Corrector) Method

This method requires value of the functions at x_i, x_{i-1}, x_{i-2} for computing the value at x_{i+1}, ie, to get the value of y_4, given y_0, the values of y_1, y_2 and y_3 are to be known.

Putting $y_4 = y_0 + \int_{x_0}^{x_4} f(x,y)dx$, and $f(x,y) = f_0 + n\Delta f_0 + \frac{n(n-1)}{2}\Delta^2 f_0 + \frac{n(n-1)(n-2)}{6}\Delta^3 f_0 2$

(Newton's forward difference formula)

$$
\begin{aligned}
y_4 &= y_0 + \int_{x_0}^{x_4} \left[f_0 + n\Delta f_0 + \frac{n(n-1)}{2}\Delta^2 f_0 \ldots\ldots \right] dx \\
&= y_0 + h\int_0^4 \left[f_0 + n\Delta f_0 + \frac{n(n-1)}{2}\Delta^2 f_0 \ldots\ldots \right] dx \\
&= y_0 + h\left[4f_0 + 8\Delta f_0 + \frac{2D}{3}\Delta^2 f_0 + \frac{8}{3}\Delta^3 f_0 + \ldots\ldots \right] \\
&= y_0 + \frac{4h}{3}[2f_1 - f_2 + 2f_3]
\end{aligned}
$$

The above is called PREDICTOR VALUE of y_4 wherein

$$
\begin{aligned}
& f_0 = f(x_0, y_0) \\
& f_1 = f(x_0 + h, y_1) = f(x_1, y_1) \\
& f_2 = f(x_0 + 2h, y_2) = f(x_2, y_2) \\
& f_3 = f(x_0 + 3h, y_3) = f(x_3, y_3) \\
\text{and}\quad & y_1 = f(x_1) = f(x_0 + h) \\
& y_2 = f(x_2) = f(x_0 + 2h), \\
\text{and}\quad & y_3 = f(x_3) = f(x_0 + 3h)
\end{aligned}
$$

(These may be obtained by EULER's or any other method)
Using $y_4 = y_0 + \frac{4h}{3}[2f_1 - f_2 + 2f_3]$
f_4 is obtained as $f(x_0 + 4h, y_4) = f(x_4, y_4)$
This value of f_4 is used in the corrector formula; which is derived as follows:

$$
\begin{aligned}
y_2 &= y_0 + \int_{x_0}^{x_2} f(x, y)dx \\
&= y_0 + h\int_0^2 \left[f_0 + n\Delta f_0 + \frac{n(n-1)}{2}\Delta^2 f_0 + \ldots\ldots \right] dx \\
&= y_0 + h\left[2f_0 + 2\Delta f_0 + \frac{1}{3}\Delta^2 f_0 + \ldots\ldots \right] \\
&= y_0 + \frac{h}{3}[f_0 + 4f_1 + f_2]
\end{aligned}
$$

Similarly, it can be derived as

$$
\begin{aligned}
& y_3 = y_1 + \frac{h}{3}[f_1 + 4f_2 + f_3] \\
\text{and}\quad & y_4 = y_2 + \frac{h}{3}[f_2 + 4f_3 + f_4]
\end{aligned}
$$

So, the predicted value of y_4, say

$$
Py_4 = y_0 + \frac{4h}{3}[2f_1 - f_2 + 2f_3]
$$

and the corrected value, say

$$Cy_4 = y_2 + \frac{h}{3}[f_2 + 4f_3 + f_4]$$

where $f_4 = f(x + 4h, Py_4) = f(x_4, Py_4)$

EXAMPLE 21.7 Given $y' = x + y$, and $y(0) = 1$, find the value of $y(0.4)$

Solution:

The problem can be solved by taking $h = 0.1, x_0 = 0, y_0 = 1$

$$\begin{aligned}\text{and}\quad x_1 &= x_0 + h = 0.1\\ x_2 &= x_0 + 2h = 0.2\\ x_3 &= x_0 + 3h = 0.3\\ \text{and}\quad x_4 &= x_0 + 4h = 0.4\\ \frac{dy}{dx} &= y' = f(x, y) = x + y\\ f(x_o, y_o) &= x_o + y_o = 0 + 1 = 1 = f_o.\end{aligned}$$

Using EULER's method the values of y_1, y_2, y_3 corresponding to x_1, x_2, x_3 can be obtained as follows

$$\begin{aligned}y_1 &= y_0 + hf(x_0, y_0) = 1 + 0.1 \times 1 = 1.1\\ f_1 &= f(x_1, y_1) = 0.1 + 1.1 = 1.2\\ y_2 &= y_1 + hf(x_1, y_1) = 1.1 + 0.1(1.2) = 1.22\\ f_2 &= f(x_2, y_2) = 0.2 + 1.22 = 1.42\\ y_3 &= y_2 + hf(x_2, y_2) = 1.22 + 0.1(1.42) = 1.362\\ f_3 &= f(x_3, y_3) = (0.3 + 1.362) = 1.662\end{aligned}$$

Using the above, the Predictor value of y_4 can be obtained as :

$$Py_4 = y_0 + \frac{4h}{3}(2f_1 - f_2 + 2f_3) = 1 + \frac{4 \times 0.1}{3}(2 \times 1.2 - 1.42 + 2 \times 1.662) = 1.5739$$

$$f_4 = f(x_4, Py_4) = 0.4 + 1.5739 = 1.9739$$

Using this, the corrector-value of y_4 can be obtained as:

$$Cy_4 = y_2 + \frac{h}{3}(f_2 + 4f_3 + f_4) = 1.22 + \frac{0.1}{3}(1.42 + 4 \times 1.662 + 1.9739) = 1.55473$$

$$y(0.4) = 1.55473$$

Summary

Differential equations cannot be solved as such by the digital computers. So, numerical methods have to be evolved for their solution through computers. Depending on the numerical method, the problems involving differential equations are classified as Initial value problems and Boundary value problems.

Initial value problems may be solved by successive approximation methods or the Predictor-Corrector methods.

The successive approximation methods are the Euler's method, Runge-Kutta's second order and fourth order methods and Taylor's Series method.

The Predictor-Corrector methods are the Euler's modified method and Milne's method. These methods are found to be more accurate.

Exercise/Questions

21.1 Solve the differential equation $y' = x + y^2$, given that $y(0) = 1$, by PICARD's method and find the value of $y(0.1)$.

21.2 Using EULER's method find the value of $y(1.0)$ given that $y' = x + y$ and $y(0) = 0.5$

21.3 Given $y' = y - x$, when $y(0) = 0.5$, find $y(0.5)$ and $y(1.0)$ using RUNGE-KUTTA's SECOND ORDER method.

21.4 Solve the above problem by RUNGE-KUTTA's FOURTH ORDER method.

21.5 Given that $y' = x^2 y - 1$ and $y(0) = .1$, find $y(0.03)$ using TAYLOR's expansion

21.6 Given $y' = x^2 + y$ and $y(0) = 0.1$, find $y(0.1)$ by Modified EULER's method.

21.7 Given that $y' = x - y^2$ and $y(0) = 1$, find $y(0.4)$ by the use of MILNE's method

21.8 Write down algorithms and FORTRAN programs for all the numerical methods discussed in this chapter.

for NOTES

"This page is Intentionally Left Blank"

Part D
Computer Languages

Chapter Twentytwo	Overview of Computer Languages and Their Distinctive Features *245 – 250*
Chapter Twentythree	FORTRAN *251 – 255*
Chapter Twentyfour	ALGOL *256 – 261*
Chapter Twentyfive	COBOL *262 – 272*
Chapter Twentysix	PL/1 *273 – 279*
Chapter Twentyseven	Translators and Compilers *280 – 284*
Chapter Twentyeight	Multi Processing, Multi Programming and Time Sharing *285 – 287*

"This page is Intentionally Left Blank"

Chapter Twentytwo

Overview of Computer Languages and Their Distinctive Features

22.1 INTRODUCTION

COMPUTER LANGUAGES are better known as PROGRAMMING LANGUAGES. They are the languages in which a computer program can be written. PROGRAM is a set of unambiguous instructions in proper sequence for solving a problem with the use of computers. The rules specifying the method of constructing the instructions or statements are called the SYNTAX of the language. The meaning or interpretation of the instructions (statements) is called SEMANTICS.

The syntax of an Arithmetic statement is an IDENTIFIER followed by an assignment symbol (=) which is followed by an Arithmetic expression. (refer Fig 22.1)

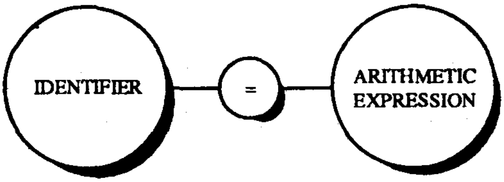


Fig 22.1 Syntax Diagram of Arithmetic Statement

$A = B + C$ is an example of such an Arithmetic Statement.

The meaning of this Statement is: Compute the value of $B + C$ and assign the value to A. This is called SEMANTICS.

22.2 TYPES OF SEMANTICS

There are three major methods of specification of Semantics (1) INTERPRETIVE Semantics, (2) TRANSLATION semantics, and (3) AXIOMATIC DEFINITION semantics.

An interpretive semantics is also called 'operational semantics'. It is defined by the computer itself.

A translation semantics is one that can be translated into interpretive semantics.

The axiomatic definition semantics defines rules that relate the data before and after execution of each program construct (instruction/statement).

22.3 TYPES OF PROGRAMMING LANGUAGES

The programming languages are of 2 major types (1) LOW LEVEL languages, and (2) HIGH LEVEL languages.

A program in low level language is made of a number of instructions, each instruction involving only one operation. A program in high level language is made up of a number of statements, each statement may generate one or more operations.

22.4 MACHINE LANGUAGE

The low level languages are of 2 types, namely (1) MACHINE language, and (2) ASSEMBLY language.

The instructions in machine language are of INTERPRETIVE semantics. Machine language is also called MACHINE CODE. Any computer can understand and execute instructions only in machine code. A program in machine code is called the OBJECT PROGRAM. Each type of computer is designed to execute programs in a machine code. Different types of computers will have different machine codes.

An instruction in machine code contains an OPCODE (Operation code/INSTRUCTION) and one or more addresses. These will normally be as stored in BINARY FORM.

OPCODE	ADDRESS 1	ADDRESS 2
5A	6	600

Above is an instruction in Hexadecimal form which means 'Add data in memory location 600 with data in the general purpose register 6'.

This will be stored in the binary form as follows

0101 1010 0110 011000000000

The computer can understand and execute an instruction in this form only. But writing instruction in machine code is difficult since it involves knowledge of memory or register addresses in which the intermediate results are stored. So, assembly languages and high level languages have been developed for program-writing.

22.5 ASSEMBLY LANGUAGE

The instructions in ASSEMBLY LANGUAGE are of translation semantics since each instruction in Assembly language can be translated to an equivalent instruction in machine language. An instruction in Assembly language contains a mnemonic and one or more symbolic operands (eg)

MNEMONIC	OPERAND 1	OPERAND 2
A	X	Y

The above instruction means 'add data in memory location of symbolic address *x* with data in memory locations of symbolic address *y*.'

Writing programs in Assembly language is comparatively easier than writing in machine code. Systems programs are often written in Assembly language.

A program in Assembly language has to be translated into Machine language for processing the data with it. For this, a systems program known as ASSEMBLER has to be used.

22.6 HIGH LEVEL LANGUAGES

High level languages are used for writing 'USER'S' programs. A program in Assembly language or in any high level language is called a SOURCE PROGRAM. A program in a high level language is made up of a number of statements. These statements are of axiomatic definition semantics. A program in high level language is also to be translated into machine language, for execution. The translator programs used in this case are called COMPILERS. Different compilers have to be used for translating source programs in different languages into the same machine code. eg, For programs in FORTRAN, FORTRAN-COMPILER is to be used and for programs in COBOL, COBOL-COMPILERS are to be used. High level languages are sometimes called 'COMPILER LANGUAGES'.

22.7 CLASSIFICATION OF HIGH LEVEL LANGUAGES

The high level languages can be classified as (i) PROCEDURE ORIENTED LANGUAGES, and (ii) PROBLEM ORIENTED LANGUAGES.

The data-base languages are problem oriented languages.

The procedure oriented languages are of 3 types, (1) Languages for processing scientific and technical data, eg, FORTRAN, (2) Languages for processing commercial and business data, (eg) COBOL, and (3) Languages that can be used for processing both commercial and technical data, (eg) BASIC, PASCAL.

22.8 FEATURES OF HIGH LEVEL LANGUAGES

(i) Besides numeric data, strings, Boolean data etc can be processed using a high level language. Boolean data is one which has 2 values only, namely, 1 (TRUE) and 0 (FALSE). This will be useful in LOGICAL operations.

(ii) High level languages have provision for structuring data, eg, in COBOL records can be described as part of file and fields as parts of records.

(iii) ALLOCATION OF STORAGE can be controlled by Programmer, eg, use of COMMON AND EQUIVALENCE statements in FORTRAN, and

(iv) FUNCTIONAL MODULARITY is possible.

ie, Program can be written in modules (in the form of sub-programs). As such, any change in a module will not affect the overall program.

22.9 ADVANTAGES IN HIGH LEVEL LANGUAGES

(1) **Machine independence:** The programmer need not be aware of the type of computer which is to compile and run the program. While Machine language and Assembly language are machine-oriented, (they are different for different types of processors), high level language programs are machine-independent. eg, A programe in FORTRAN will be the same whatever, be the computer that is to be used for processing. This facility is also called PORTABILITY.

(2) **Better Documentation:** Scanning through a program in high level language, one can infer the logic easily. Also, the provision of comments within the program makes this all the more easier. This may not be possible if the program is in a low level language.

(3) **Debugging is easier in high level language-programs:** As discused in Part B, bugs are small errors in syntax which will be indicated in the listing which is obtained while compiling.

(4) **Learning is easier and faster:** It is easier and faster to learn a high level language like FORTRAN and COBOL than a machine language or Assembly language.

(5) **Facility to communicate with the system:** The high level language program has the facility to communicate information to the compiler about the data types (by use of type statements) and the manner of organisations of data in the input and output media (FORMAT Statements)

22.10 STRUCTURE OF A PROGRAM IN HIGH LEVEL LANGUAGE

A program in a high level language should be structured such that—

(i) It is capable of concatenation;

(ii) It is capable of selection;

(iii) It is capable of performing operations in a loop;

(iv) It is capable of linking subprograms.

Concatenation means establishing logical connection. A branch statement at the end of the program to go back to its beginning is an example. This is essential to use the same program to process a large number of similar data records in a file. Selection means performing LOGICAL OPERATIONS.

22.11 THE HIERARCHICAL STRUCTURE OF A PROGRAM

A high level language program can be represented as a 'tree structure'. It is made up of BLOCKS (eg, PL/1) or SUBPROGRAMS (PASCAL, FORTRAN). The Subprograms/Blocks are made of statements which are in turn made of Arithmetic or logical expressions. These expressions will contain names or symbols (IDENTIFIERS) and Arithmetic or relational or logical operators and/or subprogram-call-instructions. (refer Fig 22.2).

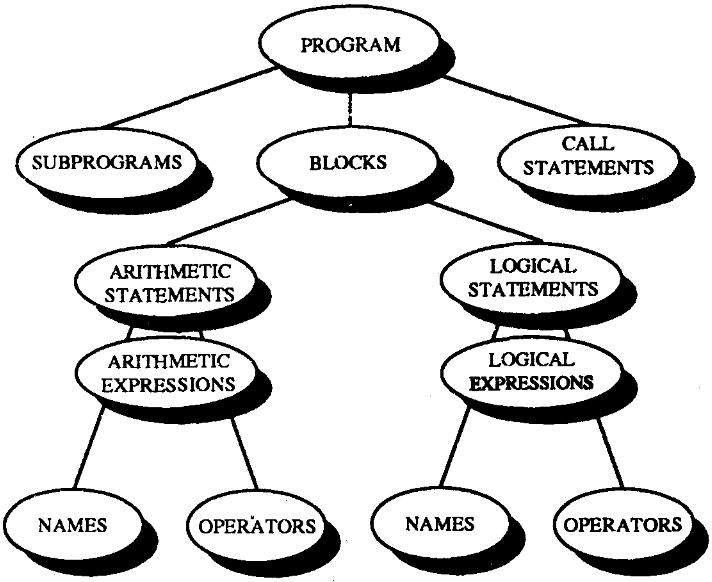

PROGRAM
SUBPROGRAMS
BLOCKS
CALL STATEMENTS
ARITHMETIC STATEMENTS
LOGICAL STATEMENTS
ARITHMETIC EXPRESSIONS
LOGICAL EXPRESSIONS
NAMES
OPERATORS
NAMES
OPERATORS

Fig 22.2

22.12 DISTINCTIVE FEATURES OF PROGRAMMING LANGUAGES

The subsequent chapters will have discussions on FORTRAN, ALGOL, COBOL and PL/1. The discussion will be on these aspects namely, (i) The character sets, (ii) The types of data, (iii) Types of possible operations and the corresponding operators, (iv) Various types of statements, their syntax and semantics, (v) Key words, (vi) Rules for the user definition of variables and constants, (vii) LIBRARY functions, (viii) SUBPROGRAMS, and (ix) APPLICATIONS. These are also called the PRAGMATICS of the languages or aspects.

Summary

Languages used for writing computer programs are called COMPUTER LANGUAGES or PROGRAMMING LANGUAGES. These are of 2 types, namely, low level languages and high level language. Low level languages are Machine language and Assembly language. Assembly language is often used for writing Systems Programs. Both the Machine language and Assembly language are machine-oriented. So the high-level languages, which are machine-independent—are used in programming. Since the computer can execute instructions in machine language only, any program in high level language has to be translated into machine code using a suitable compiler.

Exercise/Questions

22.1 Differentiate between SYNTAX and SEMANTICS, with suitable example.

22.2 What are the different types of semantics?

22.3 Explain the terms—(i) Machine Code, (ii) OPCODE, (iii) MNEMONIC, (iv) SYMBOLIC Operand, (v) Assembler, (vi) COMPILER.

22.4 Explain what is meant by ASSEMBLY LANGUAGE as also its applications.

22.5 Write short notes on—(i) Classification of high level languages, (ii) Features of high level languages, (iii) Advantages of high level languages, and (iv) Aspects of high level languages.

22.6 Describe the structure of a program in a high level language.

22.7 What are the pragmatics of a high level language?

for NOTES

Chapter Twentythree

FORTRAN

23.1 INTRODUCTION

FORTRAN stands for FORmula TRANslator. It is a language widely used for scientific and engineering computations. In fact, it was the first high-level programming language to become widely used. There are different versions of FORTRAN, say FORTRAN II, FORTRAN IV, FORTRAN 77, FORTRAN VI etc FORTRAN IV has been extensively dealt in Part B of this book. In this chapter the aspects of FORTRAN 77, which is the current standard version, is discussed.

23.2 CHARACTER SET

The set of symbols that can be used in writing the program is called the Character Set.

The Character Set of FORTRAN 77 consists of:

(1) 26 UPPER CASE ALPHABETS $(A, B, \cdots\cdots Z)$

(2) 10 numeric characters $(0, 1, 2 \cdots\cdots 9)$, and

(3) 12 special characters $[+, -, *, /, =, (,)$, COMMA, QUOTE, BLANK, $.]

COMMA =

QUOTE =

$ = dollar symbol

. = decimal point

23.3 DATA TYPES

FORTRAN 77 can operate on the following types of data:

(1) Numeric data, (2) Logical or Boolean data, (3) Character strings, (4) Arrays, and (5) Files.

The numeric data may be (1) Integer, (2) Real numbers in the Decimal form, (3) Real numbers in the Exponential form (double precision real), and (4) Complex numbers.

FORTRAN has limited facility for data-structuring which is limited to arrays and character strings of fixed length.

The numeric data can be variable or constant. If variable, it is given a name by the programmer. This name is called IDENTIFIER. It is formed from combination of maximum six alphameric characters with the first character being essentially an alphabet.

The variables may be SCALARS or ARRAYS. A scalar is a single valued variable.

A array is a set of scalars having the same 'family name'. Each element in the array is identified by the subscript, eg, A(11).

23.4 TYPES OF STATEMENTS

The statements in FORTRAN 77 can be classified as (1) I/O Statements, (2) ARITHMETIC statements, (3) CONTROL statements, (4) ITERATIVE statements, and (5) TYPE statements.

The input statement uses the verb 'READ', eg, READ A,B,C,I.

The output statement may use the verb 'PRINT' or 'WRITE'.

eg, PRINT A(I), I=1,10.

The print-out will be in the system defined format. The user may explicitly define the formats of input and output data just as in FORTRAN IV.

The Arithmetic statements in FORTRAN 77 are similar to those of FORTRAN IV, discussed in Part B.

23.5 SEQUENCE CONTROL STATEMENTS

As in FORTRAN IV, the executable statements in the programs are executed in the order of their sequence unless the sequence is modified by a control statement. The sequence control statements are (1) GOTO, and (2) CONDITIONAL statements.

23.5.1 GOTO Statements

The GOTO statements use 'STATEMENT LABELS'.

eg, GOTO 10.

FORTRAN 77 has 3 types of explicit GOTO statements. (1) UNCONDITIONAL GOTO, (2) ASSIGNED GOTO, (3) COMPUTED GOTO.

The unconditional GOTO is used to effect a transfer of control to another statement.

The assigned GOTO allows transfer to different statements depending upon the value of the label variable.

The computed GOTO allows multiway branching based on the value of an integer variable.

These have been discussed in Part B of this book.

23.5.2 Conditional Statements

The conditional statements are: (1) COMPUTED GOTO, (2) ARITHMETIC IF, (3) LOGICAL IF, and (4) BLOCK IF statements. The first three types have been discussed in Part B.

The BLOCK IF statement is similar to that in PASCAL. It is an IF statement without the use of any labels.

The form is

IF (test expression 1) THEN

— Sequence of statements.

ELSE IF (test expression 2) THEN

— Sequence of statements

—

—

—

ELSE

— Sequence of statements

END IF.

The ELSE IF or ELSE can be omitted depending on the conditions, as in PASCAL. But each of the IF, ELSE, IF, ELSE and END IF must appear on a separate line in the program because of the statement-per-line syntax of FORTRAN.

The test expressions may contain variables and relational operators and also logical variables and logical operators (These operators are the same as in FORTRAN IV)

A test expression may be of the form—

IF (A.GE.B) THEN etc

OR

IF ((A.GE.B). AND (C.EQ.D)) THEN etc.

23.6 ITERATION STATEMENTS

FORTRAN iteration statement is of single form which is called a DO statement. This statement has been discussed in all details in Part B. The only difference in the implementation of DO statement in FORTRAN IV and FORTRAN 77 is, FORTRAN IV tests the termination-condition on a DO-loop only after the first execution of the statement in the loop; so the DO loop will be executed at least once, even if the lower limit of the index variable was less than the final value specified on the first entry to the loop. FORTRAN 77 makes provision for the test to be made even before the first execution of the loop.

As in any other language, ARRAYS will be processed by DO Statements. The subscript ranges for arrays should be declared explicitly by a DIMENSION statement, eg,

DIMENSION A(100), I(20).

Another method will be to give the subscript range along with the explicit definition of variable. eg.

REAL LNGTH (80)

INTEGER SENO (200)

The use of type statements is the same as in FORTRAN IV.

23.7 SUB-PROGRAMS AND LIBRARY FUNCTIONS

Just as in FORTRAN IV, FORTRAN 77 provides for three types of subprograms: (1) ARITHMETIC statement FUNCTION, (2) FUNCTION SUBPROGRAM, and (3) SUBROUTINE SUBPROGRAM.

The rules for writing these subprograms are the same as in FORTRAN IV. The parameter transmission between the main program and the subprogram is by reference.

The list of LIBRARY functions available in FORTRAN 77 is the same as in FORTRAN IV.

23.8 CHARACTER STRINGS

FORTRAN IV can transfer a string of maximum 4 characters from input to output but cannot process string data. But FORTRAN 77 can process fixed length character-string variables. Such variables can be declared by a 'type' statement: eg,

CHARACTER A*15, B*20.

This defines A as a string of 15 characters and B as a string of 20 characters.

Character strings may also be declared as array of characters. The relational operators can be used with the strings. The result will be according to the lexicographic ordering.

eg, A.LT.B. C.GT.B.

Two strings may be concatenated into a single string. The operator used is '//' eg,

c=A//B

If A='UNDER'

and B='STAND'

C will be='UNDER STAND.'

Also a substring can be extracted from a character string using the syntax

((CHAR VARIABLE name) (FIRST CHAR POSITION) (LAST CHARACTER POSITION))

eg,

S=(A,3,6)

If A='ATTENDANCE'

S will be='TEND'.

23.9 FILE PROCESSING

Processing of files is one of the applications of FORTRAN 77.

For reading data records in a file, the READ instruction is used. But an 'OPEN' file instruction is to be used before any executable statement in the program. Similarly, after the last executable statement, a 'CLOSE' file instruction should be given before the STOP and END statements.

The Read and Write Statements may be list directed or followed by format statements. If list directed data from one record is read at a time. There should be minimum one blank in between fields in the input records. As stated earlier, the list directed write statement will give the output data in the system define format. A comprehensive READ statement may be of the form

READ (UNIT=1, FMT=20, END=100, ERR=150) A,B,C

This would mean 'Read values of variables A,B,C from input device of identification number 1, according to the FORMAT given in statement 20. If it is an END-Of-FILE condition, GOTO statement 100. If Read-error, GOTO Statement 150.

If the input or output file is on tape, the 'Back Space' instruction can be used to go back to the previous record. Also a REWIND instruction can be used at the end of processing to rewind the tape.

Since a number of FORTRAN IV programs have been given in Part B and since the FORTRAN 77 programs are similar to ones in FORTRAN IV and FORTRAN 77, compiler can compile programs in FORTRAN IV, no specimen program in FORTRAN 77 is given.

Summary

FORTRAN is a widely used high level programming language for solving technical and scientific problems. One of the recent versions of FORTRAN is FORTRAN 77. As compared to earlier versions, the additional features in FORTRAN 77 are—(1) The input-output statements may or may not have a FORMAT Statement, (2) If no format statement is given for the output statement, the output will be in the system designed format. (If format statement is given, the output will be according to this format statement), (3) STRING data can be processed, (4) Block IF statement can be used, (5) File processing can be done.

Exercise/Questions

23.1 Write down the character set of FORTRAN.

23.2 Describe the type of DATA that can be processed by FORTRAN 77.

23.3 Briefly explain the (1) 3 types of GOTO statements, and (2) 3 types of IF statements.

23.4 Bring out the difference in the execution of DO statement in FORTRAN 77 as compared to that in FORTRAN IV.

23.5 Write short notes on—(i) Character String processing, and (ii) File processing in FORTRAN 77.

23.6 What additional facilities are available in FORTRAN 77 as compared to FORTRAN IV?

23.7 Rewrite FORTRAN IV programs in Part B in FORTRAN 77.

for NOTES

Chapter Twentyfour

ALGOL

24.1 INTRODUCTION

ALGOL means ALGOrithmic Language. It is a structurally consistent, systematic, logical and powerful language. ALGOL expressions are quite similar to the corresponding mathematical representations. Also, from the flow-chart,the corresponding ALGOL program can be written systematically and easily. As such, it is often called a 'clean language'.

ALGOL has been the first language to use:= as assignment sign instead of =.

Just as FORTRAN, ALGOL also has a few versions such as ALGOL 58, ALGOL 60 and REVISED ALGOL 60. The REVISED ALGOL 60 is discussed in this chapter.

24.2 THE CHARACTER SET

The character set of ALGOL consists of

(1) UPPER CASE letters (A to Z)
(2) LOWER CASE letters (a, b, c, d, etc ... z)
(3) The ten digits (0 to 9).
(4) The Arithmetic operators: $+ - * / \div \uparrow$
(5) The relational operators: $<, \leq, >, \geq, =, \neq$
(6) Boolean operators $\equiv \vee \wedge \neg \supset$
(7) Assignment Symbol:=
(8) Delimiters . : ; ,
(9) Brackets [ ] ( )

24.3·KEYWORDS

The following is the list of KEYWORDS in ALGOL. Keywords are those that convey special meaning to the compiler and processor. As such, these cannot be used as DATA names (IDENTIFIERS)

(i) GOTO
(ii) IF THEN ELSE
(iii) FOR UNTIL
(iv) WHILE DO
(v) BEGIN, END
(vi) TRUE, FALSE
(vii) REAL, INTEGER, ARRAY, STRING, BOOLEAN
(viii) SWITCH, OWN, VALUE, LABEL, COMMENT, PROCEDURE

24.4 CONSTANTS AND VARIABLES

ALGOL can process integer and real numbers (scalars), Boolean data, arrays and strings. Examples of integer constants are

15, −20

Examples of real constants

$$18.0, 17.8, -16.2, 1.3_{10^2}(1.3 * 10^2), 0.34_{10^{-2}} \text{ etc.}$$

Variables are represented by IDENTIFIERS (names) which are alphanumeric strings beginning with a letter. There may be blanks within the name, such blanks are ignored. There is no naming convention for integers and for real numbers as in FORTRAN. But they should be defined by declaration statement in the beginning of the program, eg,

REAL XONE, XTWO, a,b,c,d

INTEGER I,N

Arrays are represented by subscripted identifiers, eg, LNGTH [10].

Rectangular brackets are to be used for enclosing subscripts. Any number of subscripts can be used. eg,

HEIGHT [1,4,8]

Array declaration is to be as follows:

REAL ARRAY BRDTH [1:100] Where 1 is the initial index and 100 the final index.

24.5 I/O STATEMENTS

Since the INPUT and OUTPUT operations deal with physical devices, their implementation in ALGOL is peculiar. For example 'x: = read' is an instruction for input operation. By this, x will take the value just read from input medium. If 2 or more values are to be read, there should be a separate 'read' instruction for each, eg,

U: = read;

V: = read;

Also each statement in an ALGOL program should be terminated with a semicolon (;).

The print statement is used for outputting the results. In this case also there should be as many print statements as the number of values to be printed. The format of the numerical data to be printed may be given in that instruction itself, eg, (i) PRINT (I,4);

This would output an INTEGER of 4 digits, (ii) PRINT (B,4,3);

This means 'print the value of REAL variable which is of 8 digits with 4 digits in the INTEGER portion and 3 on to the right of decimal point.' (Note the difference between this and F-code specification in FORTRAN).

24.6 ARITHMETIC STATEMENTS

The assignment sign used in an Arithmetic statement is: =. The rules regarding the Arithmetic statement and the order of precedence of operators are the same as in FORTRAN.

The vertical arrow (↑) is used as a sign of exponentiation in ALGOL, eg, e^2 should be written as e ↑ 2.

There are 2 operators for division, namely / and ÷. Slash ('/') is used in case of real arithmetic and ÷ is used for integer arithmetic

I := J ÷ K;

I will be the truncated integer QUOTIENT.

In case of real-Arithmetic the results are rounded-off in ALGOL (and not truncated as in FORTRAN).

Using a single arithmetic statement the same value can be assigned to any number of variables in ALGOL, eg,

a := b := c := 13.28;

24.7 RELATIONAL AND BOOLEAN OPERATORS

ALGOL uses the following symbols as relational operators:

OPERATION	OPERATOR IN ALGOL	OPERATOR IN FORTRAN
EQUAL TO	=	. EQ .
NOT EQUAL TO	≠	. NE .
GREATER THAN	>	. GT .
GREATER OR EQUAL	≥	. GE .
LESS THAN	<	. LT .
LESS THAN OR EQUAL	≤	

Following is the list of BOOLEAN operators:

OPERATION	OPERATOR IN ALGOL	OPERATOR IN FORTRAN
NOT	⌉	. NOT .
AND	∧	. AND .
OR	∨	. OR .
IMPLICATION	⊃	—
EQUIVALENCE	≡	—

(The Boolean operations in the above Table are according to their order of precedence).

A logical expression may contain all the three types of operators, namely, Arithmetic, Relational and Boolean. The precedence of execution will be (i) executions of Arithmetic operations, (ii) execution of relational operations, and finally (iii) execution of Boolean operations.

24.8 CONTROL STATEMENTS

As in any other language, statements in an ALGOL program are executed in sequential order unless specified otherwise by transfer of control statements.

Unconditional transfer of control is effected by a 'GOTO' statement. A label is to be used along with a GOTO statement. The label may be a single letter or word followed by a COLON (:), eg

```
L:
START:
```

A segment of a program using label and GOTO statement will be as follows—

```
L:A: = READ;
PRINT (A,4,2);
GOTO L;
```

Conditional transfer of control is effected by the IF statement. The IF statement in ALGOL should provide for 2 statements, one to be executed if the condition is satisfied and the other if it is not.

eg,

```
IF I > 0.005 then GOTO CALLN;
ELSE GOTO COMPN;
```

24.9 LOOP STATEMENTS

The loop control statement in ALGOL is:

FOR DO

It is of the form:

```
FOR<VARIABLE>:=<list>DO<STATEMENT>
```

If the increment is other than 1, it has to be mentioned as, say, FOR <VARIABLE>:=<INITIAL VALUE> STEP<INCREMENT>UNTIL<UPPER LIMIT>

DO<Statement>.

eg,

```
For I:=1 step 1 until 10 DO
a = b ↑ i;
```

There may be a group of statements under the FOR-DO loop. If so, the group or block is enclosed by a BEGIN and END statement.

Another form of FOR-DO statement is:

```
FOR<VARIABLE>:=<EXPRESSION>WHILE<CONDITION>DO<STATEMENT>
```

24.10 PROCEDURES

The sub-program in ALGOL is called 'procedure'. There are 2 types of procedures: (1) FUNCTION procedure, and (2) PROPER procedure.

The function procedure is similar to 'function sub-program' in FORTRAN. It is invoked in the main program by using the function identifier (name). The 'proper procedure' is similar to subroutine sub-program. It is called by a procedure statement in the main program.

ALGOL procedures can be called using either value parameters or variable (name) parameters. Also an ALGOL procedure can be recursive ie, it can call itself.

A model function procedure may be as follows—

```
REAL PROCEDURE POWER (*) [10];
REAL. *;
BEGIN
INTEGER: I;
FOR I:=1 step 1 UNTIL 10 DO
POWER [I]:=* ↑ I;
END;
```

24.11 LIBRARY FUNCTIONS

The following is the list of LIBRARY FUNCTION available in all versions of ALGOL

```
ABS (x)
SQRT (x)
SIN (x)
COS (x)
LN (x) (natural log)
EXP (x)
```

24.12 STRUCTURE OF ALGOL PROGRAM

Large programs in ALGOL are of modular structure. Each module or block should start with a BEGIN statement and terminate with an END statement. Each statement except the BEGIN statement should be punctuated with a semi colon(;). Comments can be written anywhere in the program. It should be preceded with the term 'COMMENT' which is separated from adjacent statement by a semicolon(;)

eg,

```
COMMENT PROGRAM FOR COMPUTING FACTORIAL;
```

An ALGOL program to compute the roots of a Quadratic equation is given below—

```
BEGIN
COMMENT EQUATION WITH REAL ROOTS ALONE ARE CONSIDERED;
REAL a,b,c,d,f,x one, x two.
COMMENT INPUT COEFFICIENTS;
a:= read
b:= read
c:= read
COMMENT COMPUTE DISCRIMINANT;
d:= SQRT (b * l - 4 * a * c);
COMMENT COMPUTE DENOMINATOR;
f:= a+a;
COMMENT COMPUTE ROOTS
x one:= (-b + d)/f;
x two:= (-b - d)/f;
COMMENT OUTPUT ROOTS;
```

```
PRINT (x one, 3,2);
PRINT (x two, 3,2);
END
```

24.13 CONCLUSION

ALGOL has its own advantages that have been discussed in the beginning. Its disadvantages are (i) lack of input output capability, (ii) it cannot handle structured data, and (iii) it cannot process string variables.

Summary

ALGOL has been designed to be as close as possible to mathematical notation. From the flow-chart or algorithm, the ALGOL program can be written easily and systematically. But it cannot handle structured data or process string data. But its subprograms which are called procedures can be recursive.

Exercise/Questions

24.1 What are the advantages of ALGOL over FORTRAN?
24.2 What are the shortcomings in ALGOL as compared to FORTRAN?
24.3 Give the list of the CHARACTER set in ALGOL.
24.4 Define the term 'RESERVE WORD'. What are the reserve words in ALGOL?
24.5 What are the types of data that ALGOL can process?
24.6 Differentiate between the operators / and ÷ in ALGOL.
24.7 Give the list of relational and Boolean operators in ALGOL.
24.8 Write short notes on—(i) RECURSION, (ii) FOR DO statement, (iii) Label.
24.9. What are the two types of procedures in ALGOL?
24.10 Write programs in ALGOL for all the examples in Part B.

for NOTES

Chapter Twentyfive

COBOL

25.1 INTRODUCTION

COBOL stands for Common Business Oriented Language. It is used for the solution of business problems like PAY-ROLL, INVENTORY, etc. The feature of this language is that it is designed to be natural (English-like) and, as such, the program itself serves as its documentation. eg, A statement in COBOL may be 'ADD DA, HRA, CCA, AND BASIC-PAY GIVING GROSS-PAY', which is self-explanatory.

25.2 STRUCTURE OF COBOL PROGRAM

A Program in COBOL is made up of four DIVISIONS—(1) IDENTIFICATION DIVISION, (2) ENVIRONMENT DIVISION, (3) DATA DIVISION, and (4) PROCEDURE DIVISION.

The Identification division gives the identification of the program, the programmer (author) and the organisation (Installation), whose data is to be processed with the use of the program.

The Environment division describes the computer to be used for compilation (source computer) and processing (object computer) as also the description of the I/O devices to be used.

The data division gives the format of input and output data as also of the intermediate results. The data division entries are to be written using INPUT-OUTPUT formats as reference.

The procedure division entries are written with the algorithm or flow-chart as reference. It contains solution to the problem in the form of statements/sentences in proper sequence.

25.3 TYPES OF SENTENCES

COBOL statements are known as sentences. (Sometimes a sentence may contain two or more statements). For writing these sentences, the CHARACTER set used is the same as in FORTRAN.

COBOL sentences are of two types: (1) IMPERATIVE SENTENCES, and (2) CONDITIONAL SENTENCES.

'OPEN INPUT-FILE', 'READ PAY-FILE', 'GOTO CALRTN', 'SUBTRACT DEDN FROM GROSS' are all examples of imperative sentence.

'IF GROSS > 5000 GOTO INCM-TAXRTN' is an example of conditional sentence.

25.4 KEY WORDS AND OPTIONAL WORDS

COBOL sentences are made up of words that are classified as (1) KEY WORDS, and (2) OPTIONAL WORDS.

Key words are words which are essential in conveying the meaning of a statement. A key word cannot be omitted.

Optional words are those that are not strictly required to convey the meaning of a statement; but the programmer may use them to improve the readability of the statement. eg, In the statement: 'IF GROSS IS GREATER THAN 10000', the words 'IS', and 'THAN' are optional and the rest are key words, since, 'IF GROSS GREATER 10000' would convey the requisite meaning. There should be minimum one blank on either side of any word. If it is the last word in the sentence, a period should immediately follow the last character of the word and the first word in the next sentence should begin after leaving minimum one blank, eg,

'ADD AMOUNT TO GROSS. b̸ MOVE GROSS TO P-GROSS'.

It should be noted that the operators +, − etc are also treated as words in COBOL and, as such, there should be minimum one blank preceding and following any operator.

25.5 CLASSIFICATION OF WORDS

The words in COBOL are classified as (1) Names, (2) OPERATORS, (3) CONSTANTS, and (4) VERBS.

25.5.1 Names

The names are sometimes called 'nouns'. There are four general categories of names in COBOL. They are (i) DATA names, (ii) PROCEDURE names, (iii) CONDITION names, and (iv) SPECIAL names.

These names are coined (framed) by the programmer. They (1) May be maximum 30 characters, (2) The first character should be a letter and others may be letters or digits, (3) Hyphen (dash) may be used in the middle of name for better readability but not blanks, and (4) They should not be reserve words.

Data names are those of input, output, files, records, and fields.

eg, IN-FILE, OUT-FILE, PAY-FILE, CARD-FILE, PRINT-FILE, IN-REC, PAY-REC, PRINT-REC, TAPE-REC, NAME, EMPNO, SALARY etc.

Procedure names are names assigned to individual paras (portions) of a program so that one procedure statement can refer to another. They are mostly paragraph titles, eg, READ-RTN, PRINT-RTN, COMPUT-RTN etc.

CONDITION names are those that go with conditional statements. eg, IF 'ALL-ELEC-HOME'. IF 'MALE', IF 'DOCTORATE', etc.

In the input records these names will be recorded in codified form, eg, '*M*' for male, '*F*' for female. These codes have to be defined in the Data division using a level number 88, eg,

88 MALE VALUE '*M*'.

SPECIAL-NAMES are names assigned by the programmer to various physical parts (peripherals) of a computer, eg, if 2 tape units are used, one of them may be named 'MASTER' and the other 'DETAIL'. These should be defined under the title 'SPECIAL-NAMES' in the ENVIRONMENT division.

25.5.2 Operators

Just as FORTRAN, COBOL has 3 types of OPERATORS (1) ARITHMETIC operators, (2) RELATIONAL operators, and (3) LOGICAL operators. The Arithmetic operators, their functions and precedence are the same as in FORTRAN. But these are used with a 'COMPUTE' VERB, eg,

COMPUTE INCMTAX = (Gross - 18000) * 15/100

As stated earlier, the operators are treated as words in COBOL and hence, there should be a blank preceding and following any operator. Just as in FORTRAN, parentheses are used to alter the hierarchy of operations.

The relational operators are > (greater than), < (less than) and = (Equal to).

eg, 'IF MRK < 35 GOTO FAILRTN.'

'IF BONUS > 10000 MOVE 10000 To PBONUS.'

'IF MRK=100 MOVE 'H' TO MRK.'

The logical operators are AND, OR, NOT. These are reserve words. These are used in statements of 'COMPOUND CONDITIONS', eg, 'IF NOT NIGHT-SHIFT OR NOON-SHIFT AND HOURS-WORKED > 8 THEN' etc.

25.5.3 Constants And Literals

Constants and literals are data created while running the program.

The literal is a data which is 'literally' stated. It is not defined by a name. It is of two types: (a) numeric literals, and (b) non-numeric literals.

A numeric literal may consist of any combination of the numerals 0 to 9. One decimal point and a minus sign may also appear in it. There should be no blank between the minus sign and the first digit. Example of numeric literals are:

$99, -184, 48.3, -52.99$ etc.

It should be noted that any numeric literal should be within the range 10^{-75} to 10^{75}.

Non-numeric literals are those that are not used for computation. It may be any term containing alphabets only or digits only or of alphanumeric characters like the HEADINGS, FIELD TITLES to be printed. They are defined by enclosing them within QUOTATION marks. The quotation marks are not part of the literal but are used only to define the literals. Any group of characters, inclusive of blanks, enclosed by quotes will be taken as non-numeric literals by the COBOL processor, eg,.

'PAY-BILL FOR JULY 1991'.

A constant may be anything that can be expressed by a combination of the characters in the given computer's character set. There are two types of constants, namely (1) NAMED CONSTANTS, and (2) FIGURATIVE CONSTANTS.

A named constant is defined in the DATA DIVISION where a data-name is assigned and its value specified, eg,

77 RATE PIC 99 VALUE 12. The value may also be accepted from CONSOLE while processing. It is used like any other data item; but its value can not be changed in the course of the program.

Figurative constants are certain named constants which have been provided with certain standard names. They are automatically recognised by COBOL and the programmer need not define them in the Data division in each program. The list of figurative constants is—

(1) ZERO, (2) ZEROS (or) ZEROES, (3) SPACE (blank), (4) SPACES, (5) ALL, (6) LOW-VALUE, and (7) HIGH-VALUE. Examples of usage of above are—

IF GROSS < 30000 MOVE ZERO I/O INCMTAX. MOVE SPACES TO TTLCTR.

77 D-ASH PIC × (120), VALUE ALL '-'.

The figuration constants are all Reserve words.

25.5.4 Verbs

Verbs normally specify an execution of an operation. There are 2 major types of verbs in COBOL: (a) PROCESSOR DIRECTING VERBS, and (b) PROGRAM VERBS.

The processor directing verbs are (1) ENTER, (2) EXIT, and (3) NOTE.

The ENTER verb enables the programmer to use other programming languages in COBOL. The EXIT verb is used to provide an end-point for a procedure that is to be executed by means of the verb PERFORM.

The NOTE verb is used for inserting comments in the source program.

eg, NOTE THAT THE FOLLOWING ROUTINE TO CHECK IF THE BONUS IS WITHIN THE PRESCRIBED LIMITS.

Just as the 'comments' in FORTRAN, these will appear in the program listing but will have no effect in the object program.

25.5.4.1 Program Verbs

Program verbs are used to specify data-processing steps. They can be classified as (1) I/O verbs, (2) Arithmetic verbs, (3) Data manipulation verbs, and (4) Sequence control verbs.

The I/O (INPUT-OUTPUT) verbs are (a) OPEN, (b) READ, (c) ACCEPT, (d) WRITE, (e) DISPLAY, and (f) CLOSE.

The OPEN verb is used to initiate the processing of one or more INPUT and/or OUTPUT files eg,

```
OPEN INPUT-MARKS-FILE AND
OUTPUT-RESULT-FILE.
```

A READ instruction directs the processor to get the next record from an input-file and make it available for processing. Provision is made for the execution of an imperative statement when the end of file is reached.

eg,

```
READ MASTER-FILE AT END GOTO END-OF-JOB-RTN.
```

The function of ACCEPT verb is to obtain a data element (low volume data) from an input-device during running of the program, eg,.

```
ACCEPT RATE FROM CONSOLE
```

The WRITE verb is used to record an output record or print a line, eg,

WRITE PRINT-RECORD FROM PRINT-IMAGE. The from option identifies the area wherein output result is available. It will be moved to the output area (print-record in this case) and written.

The DISPLAY verb is used to display small data (low-volume data) on an onput device, eg,

```
DISPLAY ERROR-MESSAGE
```

The purpose of CLOSE verb is to terminate the processing of the INPUT and OUTPUT files (that were opened), eg,

```
CLOSE INPUT-MARKS-FILE AND
OUTPUT-RESULT-FILE
```

25.5.4.2 Arithmetic Verbs

An arithmetic verb is to initiate an arithmetic operation. The arithmetic verbs in COBOL are (a) ADD, (b) SUBTRACT, (c) MULTIPLY, (d) DIVIDE, and (e) COMPUTE.

ADD is used to add two or more numeric values and to substitute the result (SUM) for the current value of an item, eg,

```
ADD BASIC, DA, HRA, CCA GIVING GROSS.
```

The SUBTRACT verb is used to specify the subtraction of one or more values from a specified value and the substitution of the result (difference) for the current value of an item, eg,

```
SUBTRACT TAX, PF, INSURANCE FROM GROSS GIVING NETT-PAY.
```

The MULTIPLY verb is used to multiply two numeric values and to substitute the result (product) for the current value of an item, eg,

```
MULTIPLY QUANTITY BY RATE GIVING COST
```

The DIVIDE verb is used to divide one numeric value by another and to use that result to replace the value of an item, eg,

```
DIVIDE BASIC BY 16 GIVING PF.
```

The COMPUTE verb is used with Arithmetic expressions to specify one or more Arithmetic operations, eg,

```
COMPUTE GROSS = BASIC + DA + HRA + CCA
COMPUTE CLOSING - STOCK = OPENING - STOCK + PURCHASE - SALES.
```

In all the arithmetic operations, the result will be displayed according to picture description giving in the DATA division. eg, If the result is 534.536 and picture description is 999.99, it will be displayed as 534.53 after truncating the extra digit. In case a rounding-off is desired, the 'ROUNDED' option may be provided in the statement, eg,

```
DIVIDE A BY B GIVING C ROUNDED
ADD A TO B GIVING D ROUNDED
SUBTRACT C FROM D GIVING E ROUNDED
MULTIPLY B WITH E GIVING F ROUNDED
COMPUTE NETT ROUNDED = BASIC + ALLOWANCE - DEDN
```

Note that the rounding-off is done on the least significant digits. In case the picture specification is less on the most significant digits, some of them will be cut-off from the result. eg, If the result is 5346.86 and the picture description is 999.99, the printed data will be 346.86. To guard agaisnt such possibilities, error routine may be provided in the program and the processor may be made to branch to such routine in case of 'SIZE ERROR', eg,

```
COMPUTE DEDN = AMOUNT/12 + INTEREST.
ON SIZE ERROR GOTO ERROR-RTN
```

25.5.4.3 Data Manipulation Verbs

The data manipulation verbs are (1) MOVE, and (2) EXAMINE/INSPECT.

The MOVE verb is used to transfer information from one data area to one or more data areas within the computer system. The source area and the destination area should have been defined with the same level number in the Data division. The exception is, data defined under level number 77 can be moved to a destination area of any level number.

In case of output-print, editing can take place concurrently with the moving. The editing is implicitly specified by the picture description of output fields.

The important edit functions are (1) recording decimal point if it is implied in the input, and (2) suppressing leading Zeros in numerical data.

If a data stored in memory is 58345 and its picture description in the output is 999.99, it will be edited and printed as 583.45.

Similarly, if a data in memeory is OO62842 and its picture description is ZZ9(5), it will be printed as b̸ b̸ 62842.

The EXAMINE verb is used to examine certain data fields. Certain versions of COBOL use the verb INSPECT instead of EXAMINE. This is left to the student for self-study.

25.5.4.4 The Sequence Control Verbs

The sequence control verbs specify the order in which various procedures in the program are to be executed. The sequence control verbs in COBOL are (1) GOTO, (2) PERFORM, (3) ALTER, and (4) STOP.

The 'GOTO' verb provides a method of departing from the normal sequences of procedures, eg, GOTO PRINT-RTN.

It is also useful to branch back to the beginning of the program after completion of processing a record, eg, 'GOTO READ-RTN'.

The PERFORM verb is similar to 'DO' in FORTRAN or 'FOX DO' in ALGOL. But it has wider application in COBOL. It provides a method of departing temporarily from the normal sequence of procedure-execution, in order to execute some other procedure, a specified number of times or until a specific condition is satisfied.

eg,

PERFORM INTRST-RTN VARYING PRINCIPAL FROM 1000 BY 100 UNTIL PRINCIPAL > 10000

PERFORM CACLN-RTN. GOTO PRINT-RTN.

Perform verb may be used to execute a number of sequential routines. In such case, it is enough if the names of the first and the last procedure in the sequence are mentioned, eg,

PERFORM READ-RTN THRU MOUERTN.

Perform verb can be used to execute the procedures non-sequentially (ie) in an order different from their sequence in the program. In such case they have to be mentioned in the 'PERFORM' statement in the order they have to be executed.

eg, PERFORM CALRTN, COMPRTN, MOURTN AND PRINT-RTN.

Th ALTER verb has its application similar to ASSIGNED GOTO in FORTRAN.

The STOP verb is equivalent to both PAUSE and STOP statements in FORTRAN. It halts the computer temporarily or finally. If it is used for temporary halt (just like PAUSE statement in FORTRAN), a literal is used along with it.

eg, STOP 99.

When processor halts on this statement, it will display this literal. Just as in the case of PAUSE statement, if the processor has to be halted on different error conditions, different literals are used with the corresponding stop statements so as to identify the type of error condition which has caused the 'halt'.

To differentiate from the other stop statements, the final 'STOP' in COBOL PROGRAM is written as 'STOP RUN'.

All the verbs discussed so far are COBOL 'RESERVE WORDS'

25.6 CODING COBOL PROGRAM

Just as FORTRAN, COBOL program has to be written in 80 column coding sheets. Columns 1–3 are for recording sheet numbers. Columns 4-6 are used to record line numbers. Line numbers are normally recorded in order of tens so that any additional step, if found necessary while reviewing, may be inserted in proper place. Column 7 is used to record 'CONTINUATION MARK' which is a '–'. From column 8 onwards the DIVISION, SECTION and PARA titles are written. From column 12 onwards up to column 72 the entries on statements are written. Columns 73 to 80 may be used to record the identification of the program and or programmer.

As stated earlier, a COBOL program is of four divisions.

In the IDENTIFICATION division, the entry against PROGRAM-ID is very important. The entry should be a single term of maximum 6 or 7 characters.

The ENVIRONMENT DIVISION has 2 sections (i) CONFIGURATION SECTION which defines the source and object computers, and (ii) INPUT-OUT section. In this section under FILE-CONTROL, the I/O devices are defined using 'SELECT' and 'ASSIGN' verbs, eg,

SELECT PAY-FILE ASSIGN TO DISK.

SELECT PRINT-FILE ASSIGN TO PRINTER.

The DATA division describes the input/output files, records and fields. For describing files the title FD is used and for records level number 01 and for fields level number 02 to 49 are used. Each field has its nature and length defined by a 'PICTURE CLASS', eg,

02 RATE PIC 99*V*99

(Rate is a numeric field of 4 characters with 2 decimal positions, decimal point not recorded).

02 NAME PIC 20(x).

Name is a string (alphameric field of 20 characters.)

02 PAY PIC 9999.99.

(Pay is a numeric field of 7 characters with 2 decimal positions; decimal point is recorded)

02 FILLER PIC X (40)

(This is the description of a blank field of 40 characters)

Unlike in FORTRAN, fields in the entire record inclusive of the blanks on the extreme right, have to be defined in the Data division.

The Data division is of 2 sections:

(1) FILE SECTION, and (2) WORKING STORAGE SECTION.

Under File Section input files, records and fields as also the output files and records are to be described.

In the Working Storage Section, results to be computed (including intermediate results) and/or to be displayed and the output fields are to be described.

The results to be computed or to be displayed are defined using a level number 77.

The PROCEDURE divisions will be of English-like statements. It gives solution to the problem.

25.7 A SPECIMEN PROGRAM IN COBOL

A firm keeps a card-file whose records are of format:

Columns 1–5	Part-number
6–20	Part-name
21–23	Quantity on stock
24–30	Rate in Rs & Ps (decimal point unrecorded)
31–80	BLANKS

Write a program in COBOL to process the above file and print the data on cards as also the cost (Rate * Quantity) with one line per material. The total cost of all materials on stock should be printed at the end.

(REFER CODING SHEETS)

Summary

COBOL is used for programming business problems.

A COBOL program is of four divisions. The IDENTIFICATION division gives the identification of the firm whose data is to be processed, the programmer and the program. The ENVIRONMENT division decribes the computer and the I/O devices. The DATA division gives the format for input and output data. The PROCEDURE division gives the processing steps in the form of sentences. The sentences are formed by COBOL words which may be OPTIONAL words or KEYWORDS. The keywords are names, operators, constants and verbs. The VERBS specify the operation to be performed.

The defect in COBOL is that it is not precise and its implementation in different system is varied.

Exercise/Questions

25.1 Briefly describe the structure of a COBOL program.

25.2 What are the two types of sentences in COBOL?

25.3 Differentiate between KEYWORDS and OPTIONAL words.

25.4 Give a list of RESERVE words in COBOL.

25.5 Describe the different types of names in the COBOL program with suitable examples.

25.6 Describe the different operators and their applications.

25.7 Differentiate between—

(a) CONSTANT and LITERAL;

(b) Numeric literal and non-numeric literal;

(c) Figurative constants and named constants.

25.8 Briefly explain the I/O verbs in COBOL.

25.9 Describe with suitable examples the Arithmetic verbs.

25.10 Give the different usage of the verb 'PERFORM'.

25.11 Write short notes on—(i) File-control; (ii) Working-storage section; (iii) GOTO statement; (iv) Stop statement; (v) ROUNDING option; (vi) MOVE verb.

for NOTES

COBOL PROGRAM SHEET

System

Program STOCK STATUS

Programer SRINIVASAN DATE: 1.7.92

Sheet 1 of 3

Identification
73] [80

SEQUENCE
PAGE SERIAL Cont.
1 3 4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
001010 IDENTIFICATION DIVISION.
   020 PROGR-ID. STSSTS.

   030 ENVIRONMENT DIVISION.
   040 CONFIGURATION SECTION.
   050 SOURCE-COMPUTER. IBM-370.
   060 OBJECT-COMPUTER. IBM-370.
   070 INPUT-OUTPTT SELECTION.
   080 FILE CONTROL.
   090        SELECT INPUT-FILE ASSIGN TO CARD-READER
   100        SELECT OUTPUT-FILE ASSIGN TO PRINTER.

   110 DATA DIVISION.
   120 FILE SECTION.
   130 FD     INPUT-FILE
   140        RECORDING MODE F
   150        LABEL RECORDS OMITTED
   160        DATA RECORD IS INPUT-REC.
   170 01     INPUT-REC. .
   180        02  PART-NOPIC 9(5).
   190        02  PART-NAME PIC X(15).
   200        02  STOCK-QTY PIC 999.
   210        02  RATE PIC 99999V99.
001220        02  FILLER PIC X(50).
```

COBOL PROGRAM SHEET

System		**Sheet 2 of 3**
Program	STOCK STATUS	Identification
Programer	SRINIVASAN DATE: 1.7.92	73] [80

SEQUENCE PAGE SERIAL Cont
1 3 4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
002    FD  OUTPUT-FILE
           RECORDING MODE  F
           LABELRECORDS  OM ITTED
           DATARECURD IS  PRINT-REC.
       01  PRINT-REC PICX(80).
       WORKING-STORAGE SECTION.
       77  COST  PIC 9(7)199.
       77  TOTAL-COST PIC  9(10)199 VALUE ZERO.
       01  PRINT-IMAGE.
           02  FILLER PIC  X(10)  VALUE SPACES.
           02  PNO PIC  9(5).
           02  FILLERPICX(5)  VALUE SPACES.
           02  PNAME PIC  X(15).
           02  FILLERPICX(5)  VALUE SPACES.
           02  PQTY PIC  999.
           02  FILLER PIC  X(5)  X(5)  VALUE  SPACES.
           02  P-RATE PIC  PIC  9(5).99.
           02  FILLER PI CX(5)  VALUE SPACES.
           02  PCOST PIC  9(7).99.
           02  FILLER PIC  X(9)  VALUESPACES.
       01  END-OF-JOB-LINE.
           02  FILLER PICX(24)  VALUE SPACES.
           02  FILLER PIC  X(20)  VALUE 'TOTAL COST OF STOCK='.
           02  PTOTAL-COST  PIC  9(10).99.
           02  FILLER PIC  X(23)  VALUE SPACES.
```

COBOL PROGRAM SHEET

System

Program STOCK STATUS

Programer SRINIVASAN DATE: 1.7.92

Sheet 3 of 3

Identification

73] [80

SEQUENCE
PAGE SERIAL Cont.

```
1   3 4   6 7 8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
003010 PROCEDURE DIVISION.
   020 BEGIN
   030 OPEN FILES.
   040     OPEN INPUT-FILE OUTPUT-FILE.
   050 READRTN.
   060     READ INPUT-FILE AT END GOTO EOFRTN.
   070     COMPUTE COST = STOCK-QTY × RATE.
   080     ADD COST TO TOTAL-COST.
   090     MOVE PART-NO TO PNO.  MOVE PART-NAME TO PNAME.
   100     MOVE STOCK-QTY TO PQTY.MOVE RATE TO P-RATE.
   110     MOVE COST TO PCOST.
   120     WRITE PRINT-REC  FROM PRINT-IMAGE AFTER ADVANCINGBY ONE LINE
   130     '  GO TO READ-RTN.

   140 EOFRTN.
   150     MOVE TTL-COST TO PTOTAL-COST.
   160     WRITE PRINT-REC FROM END-OF-JOB LINE AFTER ADVANCING BY 2 LIN
   170     ES.
   180 CLOST-FILES.
   190     CLOSE INPUT-FILE OUTPUT-FILVE
004200     STOP RUN.
```

Chapter Twentysix

PL/1

26.1 INTRODUCTION

PL/1 is an abbreviation for PROGRAMMING LANGUAGE 1. It was designed as a successor both to FORTRAN and COBOL so as to be equally useful for scientific and business applications. It had been widely used in real time and system programming applications.

26.2 CHARACTER SET

The character set of PL/1 consists of 26 upper case letters (A to Z), 10 digits (0 to 9), the dollar symbol ($), blank, the assignment symbol (=), the Arithmetic operators (+ – */) and also the following characters—

(	left parenthesis
)	right parenthesis
,	COMMA
.	period
'	QUOTE
%	percentage symbol
;	semicolon
:	colon
&	AND operator
|	OR operator
¬	NOT operator
>	Greater than
<	Less than
–	NIL
?	NIL

26.3 TYPES OF DATA

PL/1 contains an extensive selection of data types and primitives for accessing and modifying data elements. It can process (1) Numerical data, (2) Character strings, (3) Bit strings, (4) Arrays and records (structure), and (5) POINTERS.

The data type can be specified at 'COMPILE TIME' by 'Declarations'. Each declaration specifies a set of values for the various attributes which a variable or data structure will have. The computer uses these attribute declarations to determine the proper run-time representation for each data item and the way it has to be accessed and modified (eg, the declaration statement DCL (PAGENO) FIXED INIT (10); means the IDENTIFIER PAGENO denotes a fixed point (integer) number whose initial value is ZERO.

The numbers in PL/1 may be classified according to (1) mode, (2) scale, (3) base, and (4) precision. The mode of a number may be real or complex. The scale may be 'Fixed point (integer) or FLOATING POINT (real). The base may be decimal (10) or binary (2). Precision is specified by number of digits, eg,

'DCL NUM FIXED DEC (12)' specifies an integer number of 12 digits, to the base 10.

'DCL NMBR FIXED BIN (31)', specifies a binary number of integer type of 31 (binary) digits.

'DCL A FLOAT DEC (15.8)'; specifies a real number of 15 digits in all, with 8 digits after the decimal point.

The initial value of the data can also be specified in the declaration, eg, 'DCL NCOUNT FIXED DCL (10) INIT (0)'. This initiates 'NCOONT' to 0.

Character strings are the alphameric data like 'names'. The declaration for a character string should declare either a fixed length or a maximum length.

'DCL NAME CHAR (15),' specifies a fixed length character string of 15 characters.

'DCL MATLNAME CHAR (30) VAR'; specifies a character string of 'varying' length with maximum length of 30 characters.

BIT STRINGS are used in Boolean operations since there is no Boolean data type in PL/1. The basic Boolean operators AND/OR, and NOT are provided as operators on bit-strings.

Arrays and records are called as STRUCTURES in PL/1. A two dimensioned array of size 10*8 wherein each element is an integer of 5 digits will be specified as

DCL A (10, 8) FIXED (5);

In PL/1, a simple variable or an element of an array may be declared to be of type POINTER. (A pointer data is an integer number which gives the memory address of another data)

'DCL A FLOAT BASED (P)' means 'reserve an area in the memory for storing of a real number A, and assign to P, the memory address' of this area.

26.4 I/O OPERATIONS

The input and output in PL/1 are carried out through files, similar to COBOL. These files may be of 2 types: (1) STREAM files, and (2) RECORD files. Stream files are of records entered while running the program though the key-board. The record files have their records on tape, disk or card.

For stream files, GET is used for input operation and PUT for the output operation.

Both GET and PUT statements may be (1) in DATA-directed mode, or (2) in LIST directed mode, or (3) EDIT-directed mode of transmission.

For DATA directed transmission, the input file itself contains information about the identifiers to which the data are to be assigned (eg, $X = 2.5, Y = 3.8, Z = -4.2$ etc)

For output, an identifier list specifies the data to be transmitted.

In the LIST-directed mode of transmission, the input file contains values of data, which are assigned to the identifiers in the list, the reverse being the case for the output.

The EDIT-directed transmission contains the specification of the data format.

The syntax of the above will be as follows:

```
GET DATA (A,B)
GET LIST (X,Y,Z)
GET EDIT (P,Q,R) (F(3), F(6.3), A(b));
```

(The meaning of A code is the same as in FORTRAN).

```
PUT EDIT ('A=',A) (F4.1)
```

For the PUT statement there is an option to skip to a new line.

```
PUT SKIP EDIT (SUM) (F 10.2)
```

If more than 1 line has to be skipped, it has to be mentioned (eg, as)

```
PUT SKIP(2) EDIT(SUM) (F 10.2)
```

In case of RECORD-files, a READ statement is used to input a new record from the external file to an internal storage area. Similarly, a WRITE statement transmits a copy of a record on storage to the external file. REWRITE and DELETE statements are used for replacement of a record or deletion of a record, as the case may be.

26.5 ARITHMETIC AND RELATIONAL OPERATORS

The Arithmetic operations in PL/1 are the same as in FORTRAN. The precedence rules are also the same. But there is no restriction in PL/1 for mixed mode of operations.

PL/1 uses the symbols >, <, = as relational operators. Similarly, for logical operations it uses the symbols ⌉, &, |.

Arithmatic operations are possible between scalar and another scalar, between scalars and arrays as well as between arrays and arrays.

An array is specified as follows:

```
DCL (ARRAY-NAME (lower-bound: upper bound)) TYPE
```

If the lower bound is 1, it need not be specified.

26.6 CONTROL-STATEMENTS

Unconditional transfer of control (Branch) is done using GOTO Statement, eg, GOTO <label>

For conditional branching, IF statement is used. Its format is—

```
IF <CONDITION>
THEN <STATEMENT>;
ELSE <STATEMENT>;
```

eg,

```
IF A < B
THEN PUT LIST (A);
ELSE PUT LIST (B);
```

Just as in FORTRAN, LOOP Statement in PL/1 uses 'DO' The DO loop may be (a) ITERATIVE, or (b) may use a WHILE statement.

```
DO    I = 1 To 4,
DO    J = 1 To 100 By 2,
DO    X = 00 To 10 By 0.2
DO    J = 9 To 2 By -2
```

are all examples of iterative DO loops.

The form of the other type of DO loop is: DO WHILE <CONDITION>;

eg,

```
DO WHILE (N > 1);
LOOP; FACL=FACL*N
N=N-1
END LOOP:
```

Nested DO-loops are also used in PL/1.

26.7 LIBRARY FUNCTIONS

The following is the list of LIBRARY/BUILT-IN FUNCTIONS IN PL/1.

SIN(x)	x in radians
SIND(x)	x in degrees
COS(x)	x in radians
COSD(x)	x in degrees
TAN(x)	x in radians
TAND(x)	x in degrees
ATAN(x)	The arctangent of x in radians
ATAND(x)	The arctangent of x in degrees
SINH(x)	Hyperbolic sine of x
COSH(x)	Hyperbolic cosine of x
TANH(x)	Hyperbolic tangent of x
EXP(x)	e^x
LOG(x)	Natural log of x
LOG 10(x)	The base 10 logarithm
LOG 2(x)	The base 2 logarithm
ABS(P)	Absolute value of P
TRUNC(P)	Truncated value of P
ROUND (P,N)	P rounded at Nth digit
COMPLEX (P,Q)	Complex number P+iQ
IMAG(P)	The imaginary part of complex number P
REAL(P)	The real part of complex argument P
LENGTH(S)	This function returns an integer value which is the length of a charecter string S.

INDEX (STRING, PATTERN): This gives the character position in the string from wherein the pattern occurs. If the pattern is not found in the string, this returns zero.

26.8 SUB-PROGRAMS/PROCEDURES

Sub-programs in PL/1 are called PROCEDURES. These will have the usual features such as procedure name, formal and actual parameters, entry points and calling protocols. An example of a procedure is given below:

```
SWAP: PROCEDURE (x,y);
DECLARE (x,y) FIXED;
DECLARE (T) FIXED;
T=x;
x=y
y=T;
RETURN;
END SWAP;
```

This procedure will be invoked in a program by a CALL statement, eg, CALL SWAP (A,B), (A, B are the actual variables occurring in the program.)

Sub-program parameter are ordinarily transmitted by reference.

26.9 PROGRAM STRUCTURE

The program structure in PL/1 is similar to that of ALGOL. It is of block structure, ie, the program consists of one or more external blocks each of which in turn may contain one or more internal blocks.

An EXTERNAL block is one which is not contained in any other block. An internal block is one which is contained in another. A block bearing a label is delimited by a BEGIN and END statements. END statement is also used to terminate a procedure block and also DO loop.

Declarations made in a block are valid within that block and in other blocks contained within it only. So the same identifier may be used for different data elements in unrelated blocks.

Just as COBOL, PL/1 can handle structured data. In COBOL the hierarchy is indicated by two digit level numbers (01, 02 etc). Similar technique is used in PL/1, the outermost structure being indicated by the smallest number. In case any of the elements are arrays, their bounds are also indicated.

eg,

```
DCL 1 CUSTOMER (1 : 10000),
      2 NAME      CHAR(20),
      2 ADDRESS   CHAR (40)
      3 DRNO      CHAR (4)
      3 STREET    CHAR (16)
      3 LOCN      CHAR (14)
      3 PIN       FIXED (6)
```

26.10 SPECIMEN PROGRAM

Write a program in PL/1 to sum the first 11 terms in the series

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots\dots$$

The value of x is to be taken as input.

PROGRAM:

```
SUMMATION: PROCEDURE OPTIONS (MAIN)
DCL (SUM, NR, DR), FLOAT INIT(1);
DCL (x, TERM) FLOAT;
GET LIST (x);
LOOP DO I=1,10;
NR=NR*x;
DR=DR*I;
TERM=NR/DR;
SUM=SUM+TERM;
END LOOP;
PUT SKIP LIST ('SUM=', SUM);
END SUMMATION.
```

26.11 CONCLUSION

PL/1 is designed to cover a wide variety of applications. The facilities that are not available in it are graphics, simulation, formula manipulation, and pattern handling. It achieves run-time efficiency, of course, at the cost of compile-time efficiency. It can be used reasonably well in interactive, batch processing, and multi-programming environments.

Summary

PL/1 incorporates several of the features of FORTRAN, COBOL and ALGOL. The PL/1 program is modular, block structured and can be written in free-field format. It is versatile in the data types permitted and the operations available.It has almost all the facilties of an assembly language and can be used for systems programming.

Exercise/Questions

26.1 Discuss the merit and the shortcomings PL/1.

26.2 Give the list of PL/1 character set.

26.3 What are the types of data that PL/1 can handle?

26.4 Explain with examples the declaration statement.

26.5 What are STREAM FILES and RECORD FILEs? What are the I/O statements associated with each type of file?

26.6 Explain the different modes of transmission with respect to I/O statements.

26.7 Explain the 'construct' of an IF statement in PL/1.

26.8 Differentiate between iterative type DO-loop and the other type of DO-loop with examples.

26.9 Enlist the library function in PL/1 for trigonometric and logarithmic function. What are the additional functions available as compared to FORTRAN?

26.10 Write short notes on—(i) PROCEDURES, (ii) Lists, (iii) Block structure, (iv) Structured data.

26.11 For all the examples in Part B, write down programs in PL/1.

for NOTES

Chapter Twentyseven

Translators and Compilers

27.1 INTRODUCTION

The physical equipment or devices forming a computer and peripheral equipment are known as HARDWARE.

The programs and routines used to extend the capabilities of computers are known as SOFTWARE.

The software is made up of (1) USER'S PROGRAMS, and (2) SYSTEMS SOFTWARE.

The SYSTEMS SOFTWARE comprises of (a) PROGRAMMING LANGUAGES, and (b) SYSTEMS PROGRAMS.

SYSTEMS PROGRAMS are routines that extend the capabilities of the computer. The systems programs are (1) OPERATING SYSTEM, (2) FIRMWARE, (3) UTILITY PROGRAMS, and (4) LANGUAGE PROCESSORS.

The OPERATING SYSTEM is an integrated collection of service routines that performs overall supervision of sequencing of programs by a computer.

The BUILT-IN-FUNCTIONS (LIBRARY FUNCTIONS) are known as firmware.

UTILITY Programs are a group of programs for performing house-keeping chores such as sorting, merging, copying etc.

The LANGUAGE PROCESSORS are those that contribute to the processing of data by the user's programs. The language processors are (1) TRANSLATORS, (ii) LOADERS, and (iii) SIMULATORS.

TRANSLATOR is a systems program which translates the user's program (source program) from one programming language into another.

A LOADER loads the object program (in machine code) into the memory of the computer and makes it ready for execution.

A SIMULATOR which is better known as INTERPRETER is more or less a combination of translator and loader.

27.2 TRANSLATORS

There are three types of translators: (1) ASSEMBLER, (2) COMPILER and (3) PREPROCESSOR.

An ASSEMBLER translates a program in Assembly language into machine code. Each instruction in Assembly language (containing a mnemonic and one or more symbolic operands) will be translated into an OPCODE and as many absolute addresses as the number of symbolic operands. Also the instructions in machine code (object program) are assigned locations in memory sequentially.

A PREPROCESSOR is a translator that translates a program in a high level language into another high level language (eg, PASCAL to BASIC) or from a high level language into an Assembly language (eg, COBOL to AUTOCODER).

27.3 COMPILERS

A COMPILER translates a program written in a high level language (SOURCE PROGRAM) into an equivalent program in machine language (OBJECT PROGRAM). It has to select appropriate library function and subroutines as directed by the instructions or other symbols of the original routine (main program), supply the linkage that combines the functions and subroutines into a workable routine and translate tham into machine language and also link them with the main programs. The compiled routine is then ready to be loaded into memory to be run.

27.4 PHASES IN COMPILATION

The process of compilation is done in eight stages which are called phases. The eight phases are: (1) LEXICAL ANALYSIS PHASE (2) SYNTACTIC PHASE (3) INTERMEDIATE CODE GENERATION PHASE (4) MACHINE INDEPENDENT OPTIMISATION (5) STORAGE ASSIGNMENT PHASE (6) CODE GENERATION PHASE (7) MACHINE DEPENDENT OPTIMISATION PHASE, and (8) ASSEMBLY AND OUTPUT PHASE. Each phase is in the form of a module.

27.4.1 Lexical Analysis Phase

During lexical phase, the source program is scanned character by character and with the help of spaces or delimiters or operators, separated into individual entities or TOKENS.

There is a table stored along with the compiler known as 'TERMINAL TABLE'. This contains all RESERVE WORDS in the language. Each token is first checked whether it is a reserve word. Else it is checked whether it is a LITERAL. If not, it is taken as an IDENTIFIER, eg, In the statement

A=B+10,

= and + are reserve words (symbols), 10 is a literal;

A and B will be taken as IDENTIFIERS. The identifiers are entered in a Table called SYMBOL TABLE and literals in a LITERAL TABLE.

All the tokens are entered in a Table called UNIFORM SYMBOL Table and their types (RESERVE WORD or LITERAL or IDENTIFIER) are also entered along with them.

Before entering an IDENTIFIER, it is checked whether its name is according to the naming rules of the language. If not, an error message is generated to be displayed in the PRE-LIST.

27.4.2 Syntactic (Syntax Analysis) Phase

During syntax analysis, each statement in the source program is checked to see whether it is structured according to rules.

The syntax analyser module first finds out the type of statement. It ignores non-executable statements like comments. It finds whether the executable statement currently under its consideration is (a) An I/O statement, or (b) an ARITHMETIC statement, or (c) a CONTROL statement, or (d) a LOOP statement.

The compiler contains rules for each type of statement which are called GRAMMAR. These grammars are stored in the form of 'REDUCTION TABLES'.

After identifying the type of statement, the syntax analyser refers to the reduction-table and checks whether the syntax of the statement is according to the grammar. If not, an error message is generated to be displayed in the pre-list.

27.4.3 Intermediate Code Generation Phase

During this, the statements in the source program involving 2 or more operations are split into instructions of single operations and translated into an intermediate form, say into ASSEMBLY LANGUAGE, eg, An instruction.

A=B+C–D may be translated as—

```
LOAD   R1,B  (R1 is a register in the CPU)
ADD    R1,C
STORE  R1,T
LOAD   R1,T
SUB    R1,D
STORE  R1,A
```

Such translation is done using what is known as a software STACK.

The conditional statements are also similarly translated into an intermediate form.

27.4.4 Machine Independent Code Optimisation

Since the intermediate code is generated using a software stack, there is bound to be certain redundancy in it. For example, in the above example, $B + C$ is computed and the result which is available in register R_1 is sent to a memory location 'T' and again brought back to the same register for the next operation, ie, for the subtraction of D. So the above set of instructions may be optimized as

```
LOAD  R1,B
ADD   R1,C
SUB   R1,D
STORE R1,A
```

Such optimisations are necessary to reduce the memory space occupied by the object code. The optimised intermediate code will be stored in the form of a matrix.

27.4.5 Storage Assignment Phase

During this phase, each SYMBOL in the symbol table and the LITERAL in the literal table is assigned unique memory addresses. For this purpose, a LOCATION COUNTER is used. After assigning a memory address to a SYMBOL/LITERAL, the location counter is incremented by the length of data represented by the SYMBOL/LITERAL. The length is referred to from Symbol or Literal Table. It would have been entered into these tables during intermediate code generation phase with the help of FORMAT statements or data division entries in the program.

27.4.6 Object Code Generation

During this phase, the instructions in the intermediate form are converted into machine language. This is done by substituting the OPCODES for mnemonics and memory addresses for the symbolic OPERANDS.

The compiler is provided with a 'CODE DEFINITION TABLE'. From this, the OPCODE corresponding to the mnemonic is extracted.

The memory address for each symbol or literal has been entered in the symbol or literal table during the STORAGE ASSIGNMENT phase. So the addresses of the operands are taken out from these Tables and the machine code for.each instruction will be generated, eg, LOAD R_1, B may be converted into

58 01 64

where 58 is opcode, 01 register address, and 64 memory address of B, all in Hexadecimal form.

27.4.7 Machine Dependent Optimisation

This optimisation is aimed at reduction of instructions in the object program if possible. In case of large computers with 16 or more general purpose registers, the number of instructions can be reduced by using different sets of registers for different sets of operations, instead of same set of registers for all. This is called machine dependent optimisation.

27.4.8 Assembly And Output Phase

This is the final phase of the compiler. This converts the instructions in Hexadecimal form into binary form. Also it allocates memory address for each instruction, using again a LOCATION COUNTER. These addresses are referred to by LOADER while loading the object program in the memory.

Summary

Translators are among the language processors which along with OS, firmware and utilities is called a SYSTEMS PROGRAM. Besides translators, loaders and interpreters also are under the category of language processor. The translators are of 3 types: (1) Asemblers, (2) Compilers, and (3) Preprocessors. Assemblers are used to translate a program in assembly language into machine code. Preprocessors translate a program from one high level language into another or into an Assembly language.

Compilers translate a program in a high level language into machine code. Different compilers have to be used for translation of different languages.

Compilation is done in eight phases. Each phase may be prepared as a separate module or some of the phases may be combined so that there are 4 or 5 modules in all.

Exercise/Questions

27.1 Define the terms: (1) HARDWARE, (2) SOFTWARE, and (3) FIRMWARE.

27.2 Differentiate between (1) Systems Software, and (2) Systems programs.

27.3 Differentiate between (1) User's programs, and (2) Utility programs.

27.4 Differentiate between (1) PREPROCESSOR, and (2) INTERPRETER.

27.5 What are the functions of (1) An ASSEMBLER, and (2) A LOADER?

27.6 What are the various phases in a compilation? Briefly explain the operations performed in each phase.

for NOTES

Chapter Twentyeight

Multi Processing, Multi Programming and Time Sharing

28.1 INTRODUCTION

The earlier computer systems were known as SINGLE USER System. Even now the PERSONAL COMPUTERS (PC's) are of single user system. The user operates such a system on an exclusive basis. If the computer is kept idle for some time, the loss is not considerable. But if big computers (mainframes) are delineated to a single user, it amounts to inadequate use of the computer and wastage of time. To avoid this, using sophisticated OPERATING SYSTEMS (OS) like the UNIX, MULTI TASKING is adopted in big computers.

28.2 MULTI-TASKING

Multi-tasking allows more than one task (Job) to be executed at the same time. The different methods of multi-tasking are: (i) BATCH-PROCESSING, (ii) MULTI-PROGRAMMING, (iii) TIME-SHARING, and (iv) MULTI-PROCESSING.

28.2.1 Batch Processing

Batch processing involves submitting to the computer a batch of jobs and getting them processed one by one without human supervision in-between. The results of the various jobs will be output in sequence. After one batch of jobs have been executed, the next batch of jobs can be presented to the computer.

The languages FORTRAN, COBOL, PL/1, and PASCAL were basically designed for Batch-Processing. This type of processing has the following drawbacks: (1) the 'turn around time' of later jobs will be more. The turn-around-time is the time interval between the submission of a job to the computer and getting the results. (2) Any error that terminates execution of a job delays the other jobs in waiting also. (3) The batch-processing environment lacks timing constraints on the job.

28.2.2 Multi Programming

Multi programming is a term given to a system that may have several processes in states of execution. The computer which has to adopt multi programming is usually provided with I/O PROCESSORS (which are also called I/O channels) to perform the I/O operations, so that the processor can concentrate on processing operations only. The I/O operations are often called I/O BOUND JOBS, and processing operations as PROCESSOR BOUND JOBS.

Multi-programming keeps many jobs in memory. When an I/O operations of a job is being performed, the processing operations of another job is being carried out. A scheduler under the operating system schedules the jobs in the memory to get a balanced mix of I/O bound and processor bound jobs. The scheduler is to see that the processor is at no time idle unnecessarily.

The disadvantage in this method is a large memory will be required to store a number of jobs at a time.

28.2.3 Time-Sharing

The simultaneous use of single processor by more than one person, each from a separate terminal, is known as time-sharing.

In the time-sharing environment, each user/terminal is allocated a 'time slice' (say 100 milliseconds) of computer time in rotation, (ie,) the processor will access each terminal for the duration of a 'time-slice', process a few insturctions and then switch over to the next terminal in sequence, in what is known as a 'Round Robin Fashion'.

Time-sharing provides facility for a large number of users to work on-line at human speeds. It would require a large secondary memory wherein jobs are originally stored and are transferred in segments (pages) to main memory when the processor accesses the related terminal. After the time-slice is over, the pages many have to be 'swapped' back to secondary memory. This type of swapping provides an imaginary large memory area which is called VIRTUAL MEMORY.

28.2.4 Multi Processing

Multi-processing means having more than one processor operating on the same memory simultaneously. Such processors are called multi processors. Some problems require carrying out the same computation on many sets of data, eg, a weather predicting program may be operating on readings taken from a number of stations. In such case, the processors constituting the multi-processor may be connected by a network.

Multi-processing can be adopted to perform sophisticated jobs like running a 'DISPLAY' continuously or to increase the speed of particular jobs, eg, air traffic control.

Multi-processing provides reliability and speed in processing. But the problem in it is that of synchronization of related processes.

28.3 OTHER DEFINITIONS

(a) DISTRIBUTED SYSTEMS

A distributed system is a collection of computers that acts as a single machine to users.

A distributed system consists of machines used in two ways (i) DEDICATED PROCESSORS, and (ii) POOL PROCESSORS.

Dedicated processors are also known as SERVERS since they manage I/O devices. The pool processors are assigned for processing. They are returned to the 'pool' after the job is over, for reassignment.

(b) PARALLEL-PROCESSING

Some of the larger problems may be broken into relatively independent parts and each may be solved by a separate processor which may be interconnected so that data and program can be transferred between them. This is called parallel-processing.

(c) REAL-TIME PROCESSING

The processing of data or information, so that the results of processing are available in time to influence the process being monitored or controlled, is called real time processing. A computer controlling a nuclear reactor is an example of a Real-time Processor.

(d) ON-LINE PROCESSING

An on-line system consist of a CPU and peripheral equipment or devices which are all under the control of the CPU. In this, the information reflecting current activity is introduced into the data processing system as soon as it occurs and gets processed. This is also called in-line processing. Most computer controlled systems are of this type of processing.

(e) INTER-ACTIVE PROCESSING

Inter-active processing provides facility for dialogue with the computer, inputting data and getting results in 'real-time', without having to wait. It provides communication between human operators and computing systems. The computerised booking at railway or air terminals are examples of inter-active processing.

Summary

Single-user application on big computers will not be economical since the system will be undertilised. So Batch-processing or multi programming or multiprocessing is adopted with such computers. In most cases, such computers are time-shared, ie, they are operated simultaneously from a number of terminals, each terminal being accessed for a fixed time-slice in rotation.

Exercise/Questions

28.1 Differentiate between (1) Single-User System, and (2) Multi-User System.

28.2 Define the terms (1) MULTI-TASKING, and (ii) BATCH-PROCESSING.

28.3 Explain the concept of TIME-SHARING.

28.4 Differentiate between (a) Multi-programming, and (b) Multi-processing.

28.5 Write short notes on—(i) Distributed system, (ii) Parallel processing, (iii) Real-time processing, (iv) On-line processing, (v) Inter-active processing.

for NOTES

"This page is Intentionally Left Blank"